UNITED STATES AIR FORCE

SUMMER RESEARCH PROGRAM -- 1993

SUMMER RESEARCH PROGRAM FINAL REPORTS

VOLUME 14

ROME LABORATORY

RESEARCH & DEVELOPMENT LABORATORIES
5800 Uplander Way
Culver City, CA 90230-6608

Program Director, RDL Gary Moore	Program Manager, AFOSR Col. Hal Rhoades
Program Manager, RDL Scott Licoscos	Program Administrator, RDL Gwendolyn Smith

Program Administrator, RDL
Johnetta Thompson

Submitted to:

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
Bolling Air Force Base
Washington, D.C.
Decmeber 1993

REPORT DOCUMENTATION PAGE

AFRL-SR-BL-TR-98-

0168

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE December, 1993	3. REPORT TYI ...ES COVERED Final

4. TITLE AND SUBTITLE USAF Summer Research Program - 1993 High School Apprenticeship Program Final Reports, Volume 14, Rome Laboratory	5. FUNDING NUMBERS
6. AUTHORS Gary Moore	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Research and Development Labs, Culver City, CA	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFOSR/NI 4040 Fairfax Dr, Suite 500 Arlington, VA 22203-1613	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES Contract Number: F4962-90-C-0076	
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for Public Release	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The United States Air Force High School Apprenticeship Program's (USAF- HSAP) purpose is to place outstanding high school students whose interests are in the areas of mathematics, engineering, and science to work in a laboratory environment. The students selected to participate in the program work in an Air Force Laboratory for a duration of 8 weeks during their summer vacation.

14. SUBJECT TERMS AIR FORCE HIGH SCHOOL APPRENTICESHIP PROGRAM, APPRENTICEDHIP, AIR FORCE RESEARCH, AIR FORCE, ENGINEERING, LABORATORIES, REPORTS, SCHOOL, STUDENT, SUMMER, UNIVERSITIES	15. NUMBER OF PAGES
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

Master Index For High School Apprentices

Ackermann, Laura
7801 Wilshire NE
La Cueva High School
Albuquerque, NM 87122-0000
Laboratory: PL/LI
Vol-Page No: 13- 5

Alexanderson, Sarah
7173 FM 1628
East Central High School
San Antonio, TX 78263-0000
Laboratory: AL/HR
Vol-Page No: 12-25

Antonson, Stephan
800 Cypress St.
Rome Catholic High School
Rome, NY 13440-0000
Laboratory: RL/IR
Vol-Page No: 14-12

Arnold, Katherine
1400 Jackson-Keller
Robert E. Lee High School
San Antonio, TX 78213-0000
Laboratory: AL/OE
Vol-Page No: 12-30

Baits, Mark
248 North Main Street
Cedarville High School
Cedarville, OH 45314-0000
Laboratory: WL/FI
Vol-Page No: 15-11

Baker, Eugenia
501 Mosely Dr.
A. Crawford Mosley High School
Lynn Haven, FL 32444-0000
Laboratory: AL/EQ
Vol-Page No: 12-19

Bakert, Jonathan
Oneida St.
Sauquoit Valley Central High S
Sauquoit, NY 13456-0000
Laboratory: RL/ER
Vol-Page No: 14- 7

Banaszak, Brian
9830 W. National Rd.
Tecumseh High School
New Carlisle, OH 45344-0000
Laboratory: WL/PO
Vol-Page No: 15-44

Barber, Jason
1000 10th St.
Floresville High School
Floresville, TX 78114-0000
Laboratory: AL/CF
Vol-Page No: 12- 8

Bautista, Jennifer
, - 0
Laboratory: WL/MN
Vol-Page No: 15-26

HSAP Participant Data

Behm, Jessica 3301 Shroyer Rd. Kettering Fairmont High School Kettering, OH 45429-0000	Laboratory: WL/ML Vol-Page No: 15-21
Berty, Sara 4524 Linden Ave. Carroll High School Dayton, OH 45432-0000	Laboratory: AL/OE Vol-Page No: 12-31
Blanchard, William , - 0	Laboratory: WL/MN Vol-Page No: 15-27
Bond, Ryan North Jackson St. Tullahoma High School Tullahoma, TN 37388-0000	Laboratory: AEDC/ Vol-Page No: 16- 1
Bowlby, Andrea Mudge Way Bedford High School Bedford, MA 1730-0000	Laboratory: PL/GP Vol-Page No: 13- 1
Brecht, Jason 5400 Chambersburg Road Wayne High Achool Huber Heights, OH 45424-0000	Laboratory: WL/FI Vol-Page No: 15-12
Brown, David 12200 Lomas Blvd. NE Manzano High School Albuquerque, NM 87112-0000	Laboratory: PL/WS Vol-Page No: 13-19
Cabral, Aaron 800 Odelia NE Albuquerque High School Albuquerque, NM 87102-0000	Laboratory: PL/SX Vol-Page No: 13-13
Camero, Lisa 2515 Navajo St. South San Antonio High School San Antonio, TX 78224-0000	Laboratory: AL/AO Vol-Page No: 12- 2
Campanile, Nicholas 2660 Dayton-Xenia Rd. Beavercreek High School Beavercreek, OH 45434-0000	Laboratory: WL/EL Vol-Page No: 15- 7

HSAP Participant Data

Carranza, Jason
505 S. Ludlow St.
Chaminade-Julienne High School
Dayton, OH 45402-0000
Laboratory: WL/AA
Vol-Page No: 15- 1

Carroll, Shawn
1400 Jackson Keller St.
Robert E. Lee High School
San Antonio, TX 78213-0000
Laboratory: AL/CF
Vol-Page No: 12- 9

Casares, Carmen
1215 N. St. Mary's
Providence High School
San Antonio, TX 78215-0000
Laboratory: AL/AO
Vol-Page No: 12- 3

Cayton, Sabrina
5005 Stahl Rd.
James Madison High School
San Antonio, TX 78247-0000
Laboratory: AL/AO
Vol-Page No: 12- 4

Chuang, Eleanore
2660 Dayton-Xenia Rd.
Beavercreek High School
Beavercreek, OH 45434-0000
Laboratory: AL/CF
Vol-Page No: 12-10

Ciomperlik, Kara
7173 FM 1628
East Central High School
San Antonio, TX 78263-0000
Laboratory: AL/OE
Vol-Page No: 12-32

Cook, Theresa
Laboratory: WL/MN
Vol-Page No: 15-28

, - 0

Cosgrove, Kathlyn
727 E. Hildebrand
Incarnate Word High School
San Antonio, TX 78284-0000
Laboratory: AL/CF
Vol-Page No: 12- 5

Dalley, Kevin
2660 Dayton-Xenia Rd.
Beavercreek High School
Beavercreek, OH 45434-0000
Laboratory: WL/AA
Vol-Page No: 15- 2

Danelo, David
25 Burwood St.
San Antonio Christian School
San Antonio, TX 78216-0000
Laboratory: AL/HR
Vol-Page No: 12-26

HSAP Participant Data

Davis, James 1000 School Ave. Rutherford High School Panama City, FL 32404-0000	Laboratory: AL/EQ Vol-Page No: 12-20
DeBrosse, Nick 3301 Shroyer Rd. Kettering Fairmont High School Kettering, OH 45429-0000	Laboratory: WL/PO Vol-Page No: 15-45
Decker, Michael 2601 Oneida St. Sauquoit Valley Central School Sauquoit, NY 13456-0000	Laboratory: RL/ER Vol-Page No: 14- 8
Deibler, Nancy , - 0	Laboratory: WL/MN Vol-Page No: 15-29
Dodsworth, Christopher 4916 National Rd. Northmont High School Clayton, OH 45315-0000	Laboratory: WL/EL Vol-Page No: 15- 8
Dominguez, Janette 114 E. Gerald Ave. Harlandale High School San Antonio, TX 78214-0000	Laboratory: AL/HR Vol-Page No: 12-27
Ellena, Brandon 711 Anita Dr. Tehachapi High School Tehachapi, CA 93561-0000	Laboratory: PL/RK Vol-Page No: 13- 9
Ethridge, Blake 7801 Wilshire Blvd. La Cueva High School Albuquerque, NM 87122-0000	Laboratory: PL/LI Vol-Page No: 13- 6
Felderman, James N. Jackson St. Tullahoma High School Tullahoma, TN 37388-0000	Laboratory: AEDC/ Vol-Page No: 16- 2
Feucht, Danny 5833 Student St. West Carrollton High School West Carrollton, OH 45418-0000	Laboratory: WL/FI Vol-Page No: 15-13

HSAP Participant Data

Finch, David 501 Niagara Ave. Colonel White High School Dayton, OH 45405-0000	Laboratory: AL/OE Vol-Page No: 12-33
Focht, Jeremy 2660 Dayton-Xenia Rd. Beavercreek High School Beavercreek, OH 45434-0000	Laboratory: WL/ML Vol-Page No: 15-22
Foley, Jennifer 2660 Dayton-Xenia Rd. Beavercreek High School Beavercreek, OH 45434-0000	Laboratory: WL/EL Vol-Page No: 15- 9
Foth, Angela 501 Mosley Dr. A. Crawford Mosley High School Lynn Haven, FL 32444-0000	Laboratory: AL/EQ Vol-Page No: 12-21
Fowler, Brendon Chenango Ave. Clinton Senior High School Clinton, NY 13323-0000	Laboratory: RL/C3 Vol-Page No: 14- 2
Garcia, Stephanie 650 Ingram Oliver Wendell Holmes San Antonio, TX 78238-0000	Laboratory: AL/AO Vol-Page No: 12- 6
Garcia, Alejandro 2515 Navajo St. South San Antonio High School San Antonio, TX 78224-0000	Laboratory: AL/CF Vol-Page No: 12-11
Garcia, Andrea 6701 Fortuna Rd. NW West Mesa High School Albuquerque, NM 87121-0000	Laboratory: PL/SX Vol-Page No: 13-14
Gavornik, Jeffrey 5110 Walzem Rd. Roosevelt High School San Antonio, TX 78239-0000	Laboratory: AL/CF Vol-Page No: 12-12
Giles, Mark 1204 Harrison Ave. Bay High School Panama City, FL 32401-0000	Laboratory: AL/EQ Vol-Page No: 12-22

HSAP Participant Data

Ginger, David 500 E. Franklin St. Centerville High School Centerville, OH 45459-0000	Laboratory: WL/ML Vol-Page No: 15-23
Gonzalez, Christopher 1400 Jackson-Keller Robert E. Lee High School San Antonio, TX 78234-0000	Laboratory: AL/OE Vol-Page No: 12-34
Gooden, Christie , - 0	Laboratory: WL/MN Vol-Page No: 15-30
Grabowski, Holly Shawsheen Rd. Andover High School Andover, MA 1810-0000	Laboratory: RL/ER Vol-Page No: 14- 9
Gurecki, David 800 Cypress St. Rome Catholic High School Rome, NY 13440-0000	Laboratory: RL/C3 Vol-Page No: 14- 1
Hanna, Melissa 1312 Utica St. Oriskany Central High School Oriskany, NY 13424-0000	Laboratory: RL/IR Vol-Page No: 14-13
Harrison, Deanna , - 0	Laboratory: WL/MN Vol-Page No: 15-31
Hartsock, David 3491 Upper Bellbrook Rd. Bellbrook High School Bellbrook, OH 45305-0000	Laboratory: WL/PO Vol-Page No: 15-46
Hayduk, Eric 800 Cypress St. Rome Catholic High School Rome, NY 13440-0000	Laboratory: RL/OC Vol-Page No: 14-16
Hemmer, Laura , - 0	Laboratory: WL/MN Vol-Page No: 15-32

HSAP Participant Data

Hill, Thuan North Jackson St. Tullahoma High School Tullahoma, TN 37388-0000	Laboratory: AEDC/ Vol-Page No: 16- 3
Hodges, Melanie 5833 Student St. West Carrollton High School West Carrollton, OH 45418-0000	Laboratory: WL/PO Vol-Page No: 15-47
Jeffcoat, Mark , - 0	Laboratory: WL/MN Vol-Page No: 15-33
Jost, Tiffany Lincoln Rd. Lincoln-Sudbury Regional High Sudbury, MA 1776-0000	Laboratory: PL/GP Vol-Page No: 13- 2
Kitty, Alexandra 3900 W. Peterson Our Lady of Good Counsel High Chicago, IL 60659-3199	Laboratory: PL/RK Vol-Page No: 13-10
Kozlowski, Peter 500 E. Franklin St. Centerville High School Centerville, OH 45459-0000	Laboratory: WL/ML Vol-Page No: 15-24
Kress, Barry , - 0	Laboratory: WL/MN Vol-Page No: 15-34
Kulesa, Joel 940 David Rd. Archbishop Alter High School Kettering, OH 45429-0000	Laboratory: WL/EL Vol-Page No: 15-10
Lormand, Bradley PO Drawer CC Rosamond High School Rosamond, CA 93560-0000	Laboratory: PL/RK Vol-Page No: 13-11
Maloof, Adam 251 Waltham St. Lexington High School Lexington, MA 2173-0000	Laboratory: RL/ER Vol-Page No: 14-10

HSAP Participant Data

Marlow, Chris 925 Dinah Shore Blvd. Franklin County High School Winchester, TN 37398-0000	Laboratory: AEDC/ Vol-Page No: 16- 4
Martin, Amy 3301 Shroyer Rd. Kettering Fairmont High School Kettering, OH 45429-0000	Laboratory: WL/FI Vol-Page No: 15-15
Matthews, Suzanne 5323 Montgomery NE Del Norte High School Albuquerque, NM 87109-0000	Laboratory: PL/SX Vol-Page No: 13-15
McEuen, Eric 800 Odelia Rd. NE Albuquerque High School Albuquerque, NM 87102-0000	Laboratory: PL/VT Vol-Page No: 13-17
McGovern, Scott 3491 Upper Bellbrook Rd. Bellbrook High School Bellbrook, OH 45305-0000	Laboratory: WL/AA Vol-Page No: 15- 3
McPherson, Sandra Jefferson & Grove St. Bishop Brossart High School Alexandria, KY 41001-0000	Laboratory: WL/ML Vol-Page No: 15-25
Menge, Sean Route 294 Adirondack High School Boonnville, NY 13309-0000	Laboratory: RL/C3 Vol-Page No: 14- 3
Merrill, Benjamin 3491 Upper Bellbrook Rd. Bellbrook High School Bellbrook, OH 45305-0000	Laboratory: WL/FI Vol-Page No: 15-16
Middleton, Charles 4524 Linden Ave. Carroll High School Dayton, OH 45432-0000	Laboratory: WL/FI Vol-Page No: 15-17
Miksch, Virginia 727 E. Hildebrand Incarnate Word High School San Antonio, TX 78284-0000	Laboratory: AL/CF Vol-Page No: 12-13

HSAP Participant Data

Moore II, Elliot , - 0	Laboratory: WL/MN Vol-Page No: 15-35
Mortis, Rebecca 727 E. Hildebrand Incarnate Word High School San Antonio, TX 78284-0000	Laboratory: AL/HR Vol-Page No: 12-28
Morton, Gilbert 2001 McArthur Dr. Coffee County Central High Sch Manchester, TN 37355-0000	Laboratory: AEDC/ Vol-Page No: 16- 5
Neitzel, Laura N. St. Mary's Providence High School San Antonio, TX 78215-0000	Laboratory: AL/OE Vol-Page No: 12-35
Nguyen, Quynhtrang 5833 Student St. West Carrollton High School West Carrollton, OH 45418-0000	Laboratory: AL/CF Vol-Page No: 12-14
Nielsen, Eric 500 Turin Rd. Rome Free Academy Rome, NY 13440-0000	Laboratory: RL/C3 Vol-Page No: 14- 4
Northcutt, Chris 925 Dinah Shore Blvd. Franklin County High School Winchester, TN 37398-0000	Laboratory: AEDC/ Vol-Page No: 16- 6
Olson, Amanda 1000 School Ave. Rutherford High School Panama City, FL 32404-0000	Laboratory: AL/EQ Vol-Page No: 12-23
Ondrusek, Kimberly 7173 FM 1628 East Central High School San Antonio, TX 78263-0000	Laboratory: AL/HR Vol-Page No: 12-29
Ortiz, Benjamin 6701 Fortuna Rd. NW West Mesa High School Albuquerque, NM 87105-0000	Laboratory: PL/LI Vol-Page No: 13- 7

HSAP Participant Data

Page, Melissa 501 Mosley Dr. A. Crawford Mosley Lynn Haven, FL 32444-5609	Laboratory: WL/FI Vol-Page No: 15-18
Panara, Michael 500 Turin St. Rome Free Academy Rome, NY 13440-0000	Laboratory: RL/C3 Vol-Page No: 14- 5
Penn, Alexander , - 0	Laboratory: WL/MN Vol-Page No: 15-36
Perry, Kyle Crestview High School , - 0	Laboratory: WL/MN Vol-Page No: 15-37
Pletcher, Mary , - 0	Laboratory: WL/MN Vol-Page No: 15-38
Pletl, Anne Burrstone Rd. Notre Dame Utica, NY 13502-0000	Laboratory: RL/C3 Vol-Page No: 14- 6
Prevost, Daniel 3301 Shroyer Rd. Kettering Fairmont High School Kettering, OH 45429-0000	Laboratory: WL/PO Vol-Page No: 15-48
Price, Kristy North Jackson St. Tullahoma High School Tullahoma, TN 37388-0000	Laboratory: AEDC/ Vol-Page No: 16- 7
Protz, Christopher 501 Mosley Dr. A. Crawford Mosley High School Lynn Haven, FL 32444-5609	Laboratory: AL/EQ Vol-Page No: 12-24
Rader, Thomas 1505 Candelaria NW Valley High School Albuquerque, NM 87107-0000	Laboratory: PL/WS Vol-Page No: 13-20

HSAP Participant Data

Ray, Kristopher 401 Eagle Blvd. Shelbyville Central High Schoo Shelbyville, TN 37160-0000	Laboratory: AEDC/ Vol-Page No: 16- 8
Reed, Tracy 711 Anita Dr. Tehachapi High School Tehachapi, CA 93561-0000	Laboratory: PL/RK Vol-Page No: 13-12
Riddle, Cheryl Highway 55 Moore County High School Lynchburg, TN 37352-0000	Laboratory: AEDC/ Vol-Page No: 16- 9
Rodriguez, Luis 5400 Chambersburg Rd. Wayne High School Huber Heights, OH 45424-0000	Laboratory: AL/CF Vol-Page No: 12-15
Rosenbaum, David , - 0	Laboratory: WL/MN Vol-Page No: 15-39
Salinas, Carol 727 E. Hildebrand Incarnate Word High School San Antonio, TX 78212-0000	Laboratory: AL/CF Vol-Page No: 12-16
Schanding, Sarah 7173 FM 1628 East Central High School San Antonio, TX 78162-0000	Laboratory: AL/CF Vol-Page No: 12-17
Schatz, William 500 Turin St. Rome Free Academy Rome, NY 13440-0000	Laboratory: RL/IR Vol-Page No: 14-14
Schindler, David Drawer 1300 Los Lunas High School Los Lunas, NM 87031-0000	Laboratory: PL/LI Vol-Page No: 13- 8
Senus, Joe 500 Turin St. Rome Free Academy Rome, NY 13440-0000	Laboratory: RL/IR Vol-Page No: 14-15

HSAP Participant Data

Servaites, Jonathan 500 E. Franklin St. Centerville High School Centerville, OH 45459-0000	Laboratory: WL/PO Vol-Page No: 15-49
Shao, Min 869 Massachusetts Ave. Arlington High School Arlington, MA 2174-0000	Laboratory: PL/GP Vol-Page No: 13- 3
Simon, Ryan 701 E. Home Rd. Springfield North High School Springfield, OH 45503-0000	Laboratory: AL/OE Vol-Page No: 12-36
Smith, Adam Phillips Academy Andover, MA 1810-0000	Laboratory: PL/GP Vol-Page No: 13- 4
Solscheid, Jill 500 E. Franklin St. Centerville High School Centerville, OH 45459-0000	Laboratory: AL/OE Vol-Page No: 12-37
Spry, David 555 N. Hyatt St Tippecanoe High School Tipp City, OH 45371-0000	Laboratory: WL/PO Vol-Page No: 15-50
Starr, Jennifer 221 E. Trotwood Blvd. Trotwood Madison Sr. High Scho Trotwood, OH 45426-0000	Laboratory: WL/AA Vol-Page No: 15- 4
Strickland, Jefferey 501 Mosley Dr. A. Crawford Mosley High School Lynn Haven, FL 32444-0000	Laboratory: WL/FI Vol-Page No: 15-19
Tecumseh, Tony 5323 Montgomery NE Del Norte High School Albuquerque, NM 87110-0000	Laboratory: PL/VT Vol-Page No: 13-18
Terry, Nathan 75 Chenango Ave. Clinton High School Clinton, NY 13323-0000	Laboratory: RL/ER Vol-Page No: 14-11

Thomson, Randy
Laboratory: WL/MN
Vol-Page No: 15-40

, - 0

Triana, Zayda
727 E. HildeBrand
Incarnate Word High School
San Antonio, TX 78212-2598
Laboratory: AL/AO
Vol-Page No: 12- 7

Trossbach, Christina
Laboratory: WL/MN
Vol-Page No: 15-41

, - 0

Tseng, Miranda
3301 Shroyer Rd.
Kettering Fairmont High School
Kettering, OH 45429-0000
Laboratory: WL/FI
Vol-Page No: 15-20

Tutin, Darcie
Laboratory: WL/MN
Vol-Page No: 15-42

, - 0

Vaill, Christopher
Route 31
Vernon-Verona-Sherrill Central
Verona, NY 13478-0000
Laboratory: RL/OC
Vol-Page No: 14-17

Ward, Jon
Laboratory: WL/MN
Vol-Page No: 15-43

, - 0

Waterman, Sara
North Jackson St.
Tullahoma High School
Tullahoma, TN 37388-0000
Laboratory: AEDC/
Vol-Page No: 16-10

Weidner, Suzanne
7173 FM 1628
East Central High School
San Antonio, TX 78263-0000
Laboratory: AL/OE
Vol-Page No: 12-38

West, Johnny
2026 Stapleton Court
Belmont High School
Dayton, OH 45404-0000
Laboratory: WL/AA
Vol-Page No: 15- 5

HSAP Participant Data

Wick, Matthew 6400 Wyoming Blvd. Albuquerque Academy Albuquerque, NM 87109-0000	Laboratory: PL/WS Vol-Page No: 13-21
Williams, Scott 3511 Dayton-Xenia Rd. Beavercreek High School Beavercreek, OH 45434-0000	Laboratory: WL/AA Vol-Page No: 15- 6
Wright, Rudy 6701 Fortuna Rd. NW West Mesa High School Albuquerque, NM 87121-0000	Laboratory: PL/SX Vol-Page No: 13-16
Young, Matthew 5005 Stahl Rd. James Madison High School San Antonio, TX 78247-0000	Laboratory: AL/OE Vol-Page No: 12-39
Zimmerman, Amy 4524 Linden Ave. Carroll High School Dayton, OH 45432-0000	Laboratory: AL/CF Vol-Page No: 12-18

Rome Laboratory
Video Bulletin Board

David Gurecki

Final Report for:

High School Apprenticeship Program

Rome Laboratory

Sponsored by:

Rome Laboratory

Griffiss Air Force Base

Rome, NY

August 1993

Rome Labs Video Bulletin Board

David Gurecki

Abstract

Timely and effective communications are a necessity of the work environment. It is very important that people have knowledge of important events which directly affect their careers and lives, or just benefit them on a daily basis. Normal means of communication (fliers, bulletins, etc.) are often inadequate. This project is an attempt to develop a better means for getting pieces of information to their destinations.

The project involved setting up a Video Bulletin Board throughout one of the buildings of Rome Laboratory. This new system supplements, but does not replace, the old methods (e.g., distributing bulletins), which can take a long time for everyone to actually receive the information.

This document will describe the routines and modifications made to the original commercially developed software system. Without these modifications, the Video Bulletin Board software could not have been used for this application.

Rome Labs Video Bulletin Board

David Gurecki

Introduction:

The Rome Lab's Video Bulletin Board is the name of the project which was undertaken as a result of the need for better communications. There are many components of this system, making it very complicated. It is mainly composed of a series of television monitors and several PC computers. The television monitors are positioned throughout the building for people to look at to see the news and information. They monitors display information generated from a single source, which has to handle the information to be displayed in an organized manner. One PC Computer (the Broadcaster Station) serves as this source for displaying the information. Another PC Computer (the Creation Station) is used to generate the information to be displayed.

In operation, the two PCs run a special software package (Interactive Virtual Video) - the IVV program. The IVV program has two main parts; a scheduling program that runs on the Broadcaster Station, and a creation program that runs on the Creation Station. IVV is a miniature programming environment that has commands to display text, graphics, and numerous special effects.

The display programs that are produced by the Creation software are called "sequences". Each sequence is a list of commands that are executed on the Broadcaster Station which displays the information to the viewers. The programs can contain up to 500 commands, and can use pictures, sprites, and text displayed in different fonts. In addition to the information commands, there are utility commands to clear the screen, set the screen resolution, etc.

After the sequences have been created, they are transferred to the Broadcaster System. The Broadcaster System runs the RL Scheduled Program to play the announcements according to a schedule. This program, developed in-house, is actually a long sequence, which was written in IVV's commnad language. It is actually an extensive modification of the original Scheduler that came with IVV. The RL Scheduler takes advantage of IVV's limited variable capabilities to read schedule files on disk, and then runs the sequences according to when they are scheduled. As in the original Scheduler, there is a separate schedule for each day of the week. However, with the modified Scheduler, the sequences can now be scheduled for specific start and end dates, as well as start and end times, and play in a cyclic manner. These changes improved many of the shortcomings of the original Scheduler as described below.

When the sequences are "played", all the video output of the Broadcaster System is fed to the nine TV monitors located throughout the building in areas of heavy traffic. They are connected to the PC by a broadband cable system, similar to the cable TV system that runs into many people's homes. The Zenith televisions used in this application are special models that allow for blocking out the front button panel to insure that no one changes any of the settings. They can also be locked to power up on the same channel and volume. The TVs are turned on and off daily by digital timers.

The three major in-house tasks involved in setting up the Rome Labs Video Bulletin Board were:

1. Modifying the Broadcaster Scheduler Program,
2. Creating an Editor for the Broadcaster Schedules, and
3. Developing the capability for transferring files from the Creation station to the Broadcaster Station.

1. Modifying the Broadcaster Scheduler Program

Discussion of Problem:

When the Scheduler Sequence first came with the IVV software, it did not have the capabilities to fit the requirements of the RLVBB. Each sequence in the schedule required both a start time and a duration. When the start time was satisfied, the sequence would then be displayed continuously and repetitively until the duration was up. The Scheduler would then wait until the next sequence was scheduled to start, and play that one for the scheduled period of time. The Scheduler would also examine a specification for date, and only play the sequences if the date was between the start date and end date. The date could be left blank, in which case there was no restriction on the day it played. However, the start time or duration could not be omitted. The IVV software also did not provide for any cyclic operation. Although it could play a sequence once, there was no way to facilitate recycling the schedule back to the beginning line. When it reached the end of a day's schedule, it would simply wait for the computer's calendar to update to the next day and then begin to read its schedule.

Methodology:

The RLVBB was intended to have all the announcements cycle continuously, with each sequence playing once, and then return to rerun the first sequence after the last was completed. Because of the inadequacy of the original Scheduler program to perform the required tasks, several modifications were made to the coding of the Scheduler to accommodate the requirements. Here is a summary of the modifications:

1. A modification was made to allow the scheduler to play each sequence or list of sequences in a schedule only **once**, and then loop back to the beginning when it reaches the last sequence in the schedule.

2. The second modification was to give the RL Scheduler the ability to understand and accept a blank time specification. This condition is interpreted by the RL Scheduler as a

requirement to display a sequence immediately and not wait for any start time.

3. A third modification to the original scheduler's script gave the RL Scheduler the ability to process a start time and duration field in a different manner. Instead of playing a sequence repeatedly for a specified duration of time as the original code did, the new scheduler uses this parameter field to begin incorporating the sequence in cycle at the *start time*, and continues including it in the daily sequence cycle for the *duration* period specified. If no duration is specified, the sequence will be included in the cycle from the *start time* until the end of the day, at which time a new schedule is read.

4. Prior to modification, when the Broadcaster system displayed the sequences, it read through the schedule, played the sequences found on each line, and then cycled to the beginning. The fourth modification causes the Broadcaster to display the list of sequences in progress and their descriptions on the screen in between each cycle. The qualifications for date and time do not change (i. e., a sequence will not show up in the list if its time and date requirements are not met). This modification also allows the displaying of the sequence progression at the bottom of the screen. The sequence progression is a message containing where out of the total number of sequences the current sequence occurs, and what the total is. The message looks like "Seq 3 of 5". It will change for each sequence that plays in the cycle.

2. Creating an editor for the Broadcaster's Schedules

Discussion of Problem:

The schedule files that the RL Scheduler reads consist of plain DOS ASCII text files, which can be edited using a simple text editor. However, editing schedule files in this manner can be a very tedious exercise when it is necessary to add a sequence to every schedule. Also, editing files in this manner is not very user-friendly. This utility was created to make things easier for the operator to manage the schedules.

Methodology:

The Broadcaster Schedule Setup Utility (BSSU) was created in-house specifically for the Rome Lab's Video Bulletin Board. It greatly improves the user's ability to maintain the schedules by providing an interactive user interface. It allows the user to easily add, edit, or remove new sequences to any point in a schedule. There is also a global update function that allows for automatic adding or removal of a sequence in each schedule of the week.

The Setup program consists of 2,704 lines of code. It is subdivided into a total of 10 sequences, since each sequence can only contain up to 500 lines. The following is a list of the sequences and what they do:

1. *Setup* Declares variables, contains main menu, and branches off to other sections of

the program. This is the sequence that is initially loaded when the user starts the BSSU.

2. *Set_read* Reads and displays a specified schedule file on the screen. This is utilized whenever it is necessary for the user to select a line from a schedule. The other parts of the program that use this subroutine are: add, edit, remove, and view the schedules.

3. *Set_add* Adds a new sequence to the schedule. This calls *set_add1* to input the sequence parameters. It then asks which day to add the sequence to, and prompts for the line to insert it before. It will then update that particular schedule with the new sequences.

4. *Set_add1* Contains routines to input the name, times, dates, and description of the announcement. It is a subset of *Set_add*.

5. *Set_addg* Contains the routines to "Globally" update every schedule. It prompts for a model schedule, and then asks which "model "line to use to insert the new sequence before in every schedule. It checks each schedule to make sure the chosen "model" line is present in each schedule, and then adds the new sequence to each schedule automatically.

6. *Set_edit* Allows for editing of the sequences in the schedules. It uses *Set_ed1* to select a day to edit, then it prompts for a line to edit. It allows editing of each field of that line, and then replaces the new line over the original line.

7. *Set_ed1* Selects the day(s) of the schedule from which to edit a sequence.

8. *Set_remv* Allows for removing a sequence from a schedule. It prompts for which days to edit, and then displays the schedules from those days. The user can select a line to delete from the first schedule selected. It will delete the line, and then allow the user to delete another line, until they select "Done". It will then move on to the next schedule to allow the user to delete a line.

9. *Set_remg* (Global-remove sequence) Prompts the user to select a model day, and then which sequence to remove. It checks each schedule for that line, and then deletes it from each schedule automatically.

10. *Set_view* Allows the user to view any of the schedules on the screen, but does no editing.

3. Developing the capability for transferring files from the Creation station to the Broadcaster Station

Discussion of Problem:

Since the Broadcaster is always displaying the sequences, another station is required for the creation. Because these two stations are separate, new information has to be sent to the Broadcaster system in order to take effect. Originally, the way this was done was to put the updated information on a floppy disk, stop the Scheduler and display a "Please Wait" message, copy the information from the disk onto the Broadcaster, and then restart the Scheduler program. This was unacceptable since it meant taking the Broadcaster offline.

Methodology:

The two systems were connected together on an Ethernet LAN. The Broadcaster's drive was set up as the "V:" drive on the Creation Station. This means that when a user on the Creation Station changed to the V drive, the Broadcaster's drive would be accessed.

After this was accomplished, an addition was made to the BSSU to automatically update the Broadcaster station with the new files. The BSSU calls a batch file that uses DOS commands to do this. The DOS "*replace*" command is used to compare the Creation and Broadcaster directories for files that need to be updated. First, it copies all the schedules over, disregarding if they are newer or not. It then scans the Sprite, Text, Picture, and Sequence directories for files that (1) are newer on the Creation Station than the Broadcaster Station, or (2) exist on the Creation Station and do not exist on the Broadcaster Station. If any files in either of those categories are found, they are copied over to the Broadcaster. All the possible elements that could be used by the sequences are updated, to insure they will work properly on the Broadcaster.

The program checks before it exits, to see if the user has run the batch file. If is has been run, the program will exit, but if it hasn't, a warning message is displayed, and the user is given a chance to run the batch file. This will allow the user to update the Broadcaster whenever they have made changes to either the Schedules or the Sequences.

Conclusion and Notes:

In adition to the three major tasks previously discussed, the operator had to be trained to use the IVV software, the Graphic Image Library, and the Schedule Setup utility. This was done along side the work on the other modifications. A brief reference sheet was also created for the use and reference of the operator, who is the full time employee and will create sequences for the Broadcaster.

After all the modifications had been completed, the project was documented. A Tech

Memo has been written and is being prepared for publication. It includes information on both the Scheduler modifications and the creation of the Schedule Setup Utility. A User Manual was also created for the Schedule Setup Utiltiy.

The RL Video Bulletin Board is viewed by management as a very helpful system to be added to the work environment. It has now been made even better to operate and maintain by the modifications to the Broadcaster, the creation of the Schedule Setup program, and the automation of the updating procedure.

OBJECT-ORIENTED PARALLEL PROGRAMMING IN DERIC

Brendon P. Fowler

Summer Apprentice

Final Report for:

AFOSR Summer Research Program

Rome Laboratory

Sponsored by:

Air Force Office of Scientific Research

Griffiss Air Force Base, Rome, N.Y.

August 1993

OBJECT-ORIENTED PARALLEL PROGRAMMING IN DERIC

Brendon P. Fowler
Summer Apprentice

Abstract

The benefits of distributed computing obtained through the use of multiple computers, to a simulation was examined. In order to create this distribution, the object-oriented computer language DERIC, which was created at the Rome Laboratory, was used to make objects in different environments aware of each other, allowing for interaction between objects in different environments and an eventual decrease in the time needed to run the simulation. One of the keys to distributed simulations is synchronization, for false information will be returned if objects in different environments complete behaviors in the wrong order. The objects in the simulation are spread throughout the computers and require a binding feature to allow them to work in tandem. The need to maintain synchronization among these objects has created several interesting problems that do not exist in single environment simulations.

This summer I was introduced to the concepts of distributed parallel processing in order to write, with the help of my mentor, the manual for the setup and usage of DERIC.

OBJECT-ORIENTED PARALLEL PROGRAMMING IN DERIC

Brendon P. Fowler

Introduction

The object-oriented language DERIC was developed at the Rome Laboratory. DERIC stands for Distributed ERIC, which was a language developed earlier at Rome Laboratory and originally stood for Enhanced ROSS in Common Lisp. Both ERIC and DERIC are written in Lisp, as the original acronym for ERIC reveals. DERIC was developed to exploit natural parallelism and reduce the execution time of simulations. The use of multiple environments allows more objects to be active within the simulation at one time and will eventually allow greater processing speed to be obtained. DERIC itself utilizes the concept of object-oriented programming for simulations, with the classes all being created from the nebulous class "something," and gaining several inherent functions. From these classes objects are created to do the actual work in the simulation, with each object gaining by default the characteristics of its parent classes, building a hierarchy. This inheritance of characteristics also speeds the setup time for the simulation.

In order to gain a working knowledge of these concepts to facilitate the writing of the manual, I learned how to run Lisp and DERIC on Sun computers, and familiarized myself with a Macintosh IIsi. In order to understand how to program in DERIC it was necessary for me to learn and practice the basics of Lisp. I first took the GC Common Lisp tutorial and practiced with Macintosh Common Lisp to gain a feel of the programming style. I then practiced Franz Allegro Lisp on the Sun computers. Once I was fairly comfortable with Lisp I was able to, with the aid of my mentor, write several small simulations in ERIC that allowed me to witness how larger, distributed parallel simulations run and showed me the difficulties that can arise in even a simple simulation.

Methodology

As I progressed, I learned that in order to run the distributed simulations in DERIC, DERIC must be loaded into each of the environments that will be used in the simulation. An environment, in DERIC terminology, is a computer running a Lisp process with DERIC loaded into it. Once this is accomplished, each environment is then made aware of the other environments via a set of specialized commands. One environment is chosen as the main environment, and will be the focus of the DERIC time flow setup, which controls the simulation times of the multiple environments. This main environment is also where the output from the simulation will be sent, and where the user resides, allowing even a single person to control the multiple environments from that single console.

Before the simulation is run, the different classes and their objects are created. The "object-maker" is the specialized object that creates the other classes and objects. The "object-maker" is loaded into the environments when DERIC is first initially loaded. When the objects are created in an environment, that environment becomes their home environment. The user can either ask the object-maker of the specific environment to create the object, or simply create the object in the main environment and, because objects are mobile, move it to the other environment.

In sequential simulations running in one environment, such as those written in ERIC, only a single clock is needed to control the simulation. In DERIC, the multiple environments can be matched by multiple clocks, and thus each environment may contain a local clock in an attempt to exploit natural parallelism. The main environment, where the other environments report to, contains the master clock, or Ticker, along with its resident clock. The master clock controls the slave clocks; the clocks in the other environments. The master clock sends out the current simulation time, the slave clocks execute any events they have scheduled for that time, and return a message to the Ticker, which then advances the clock to the next simulation time, ensuring that all of the environments execute events in the proper order (Fig. 1).

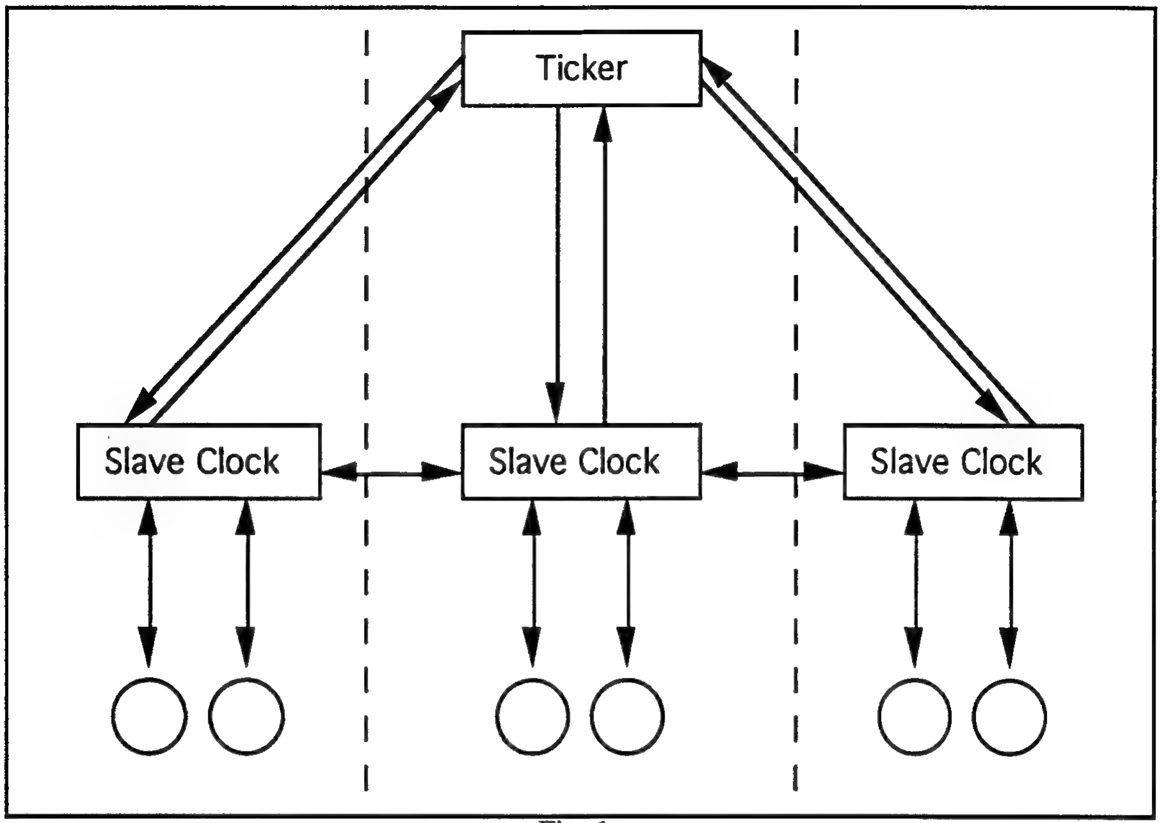


Fig. 1

In DERIC, the flow of execution through the objects is delineated by threads of control, which are sequences of atomic actions that contain identities. Threads of control are created by object requests, which are messages sent to the objects asking them to perform a behavior or action. An example of an object request would be a slave clock asking an object to perform a behavior at the given simulation time. Subthreads are threads that are created by an object receiving a thread from another object.

For example, in Fig. 2 thread t1 was created by a request to object O1. Object O1 then made two requests, one to object O4, spawning subthread t1.1, and the other to object O2, which continues the execution of thread t1 (Fig. 2).

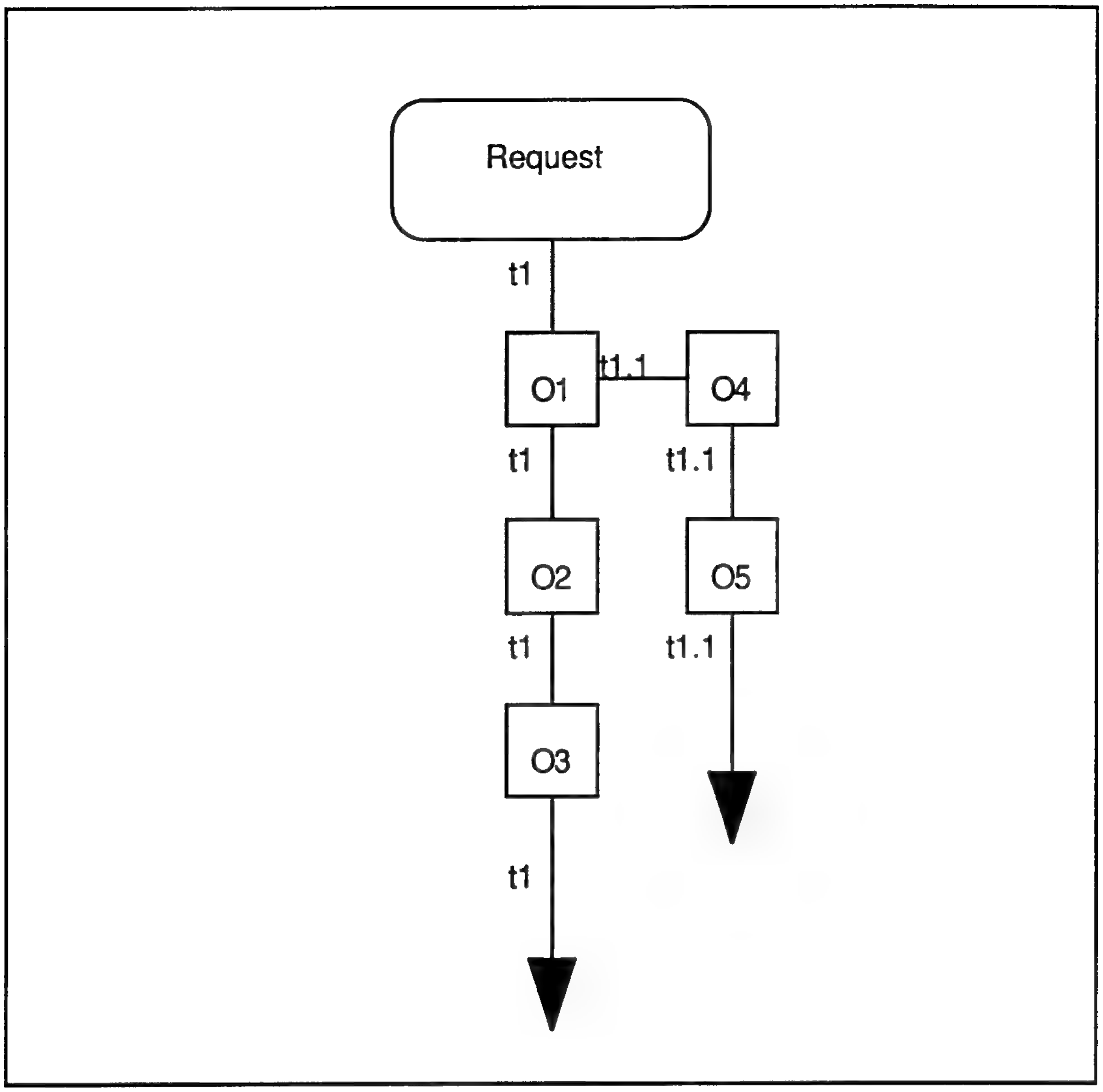


Fig. 2

<u>Results</u>

The use of DERIC, while greatly facilitating the development of distributed parallel simulations and retaining the ability of a single user to control the simulation, carries with it several problems not experienced in ERIC. Because ERIC is a sequential language, it is impossible for deadlock to occur. In DERIC, however, deadlock is a very real and common threat. In DERIC there are multiple threads in the simulation environment, a result of the various objects in the different environments making requests of each other. In order to ensure that casuality is maintained, threads "seize" objects when they request behaviors of them, blocking out all other threads until the behavior is completed. Therefore, deadlock can easily occur when two objects are seized by different threads at the same time, and then proceed to make requests of each other. These requests are not filled because the objects are still seized by original threads(Fig. 3).

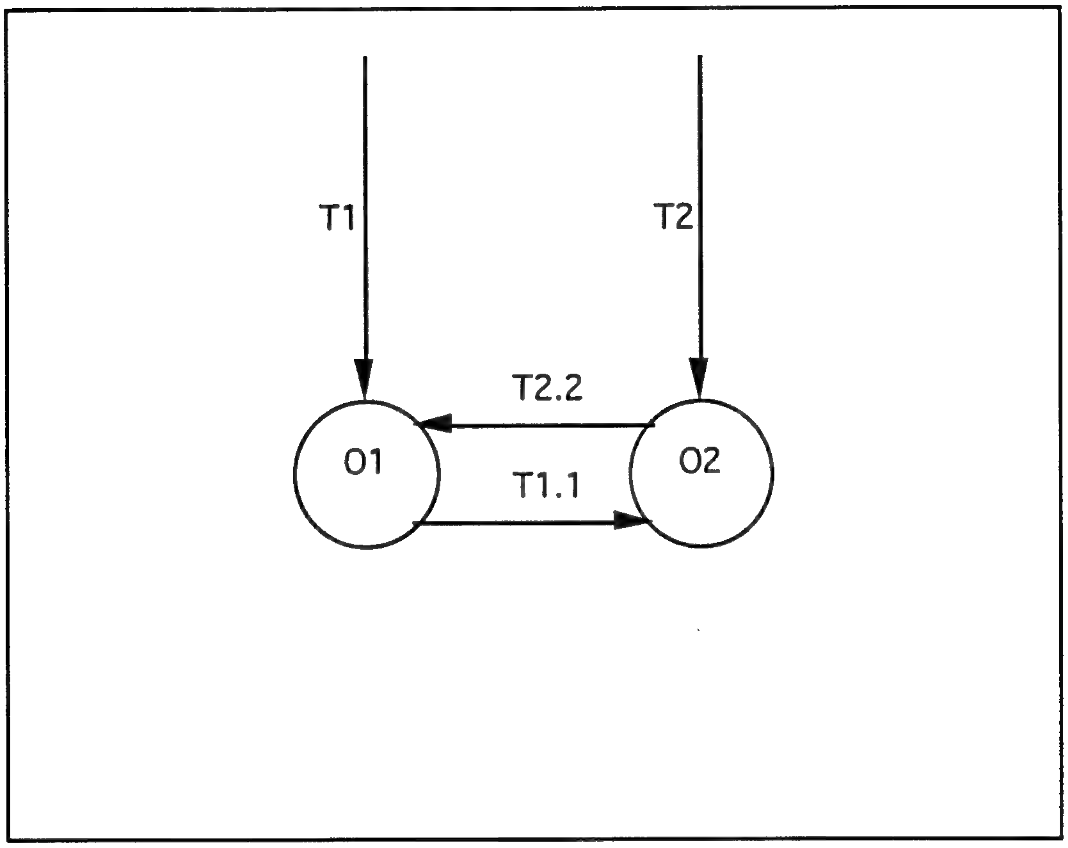


Fig. 3

At this point the simulation is deadlocked and will not progress without aid from the user.

In order to deal with deadlock DERIC was developed to allow the user to stop the simulation at any point to make changes or break deadlock, and also includes special "atomicity levels", which range from allowing all threads to seize the object to allowing no threads while the object is blocked. These "atomicity levels" are a gamble, for the levels that allow no threads to seize the object greatly restrict the capabilities of the simulation, while the level that allows all threads to seize the object is very dangerous, for false output is easily obtained.

Conclusions

The use of distribution in computer simulations is a great boon, for it allows the greater total memory of multiple environments to be harnessed and enables the simulations to become more realistic through communication between objects in different environments. Through both writing simulations and studying those written for the Air Force to simulate flight missions, I have seen how distributed simulations, written in code designed to take advantage of parallelism, are both effective and realistic, returning information to the user that facilitates improving the simulation and further increasing its realism. Although the simulations do have their difficulties, they are an improvement over single environment simulations and have vastly expanded capabilities. An interesting side note is that at this point the parallel simulations written in DERIC and executed on multiple environments are in fact often slower than an equivalent simulation on a single machine because of the high communication costs; this is one of the areas where research is being focused in order to achieve speedups.

My contributions this summer to the development of the DERIC manual have both enabled me to learn about the various computer systems and languages used in distributed simulations and aid the Air Force in documenting an important language for use in later distributed simulations. The manual will eventually culminate in an in-house report at Rome Laboratory.

Object-Oriented Systems Analysis

Sean Menge
Apprentice
Scott Shyne (mentor)
Computer Scientist

Final Report for:
High School Apprentinceship Program
Rome Laboratory

Sponsored by:
Air Force Office of Scientific Research
Boiling Air Force Base, Washington, D.C.

August 1993

Object-Oriented Systems Analysis

Sean A. Menge
Rome Laboratory
Griffiss Air Force Base, Rome N.Y.

Abstract

An in-depth look at the up and coming and very successful object-oriented analysis methodology was conducted at Rome Laboratory in order to determine an appropriate technique for software analysis of a future D.D development effort. Two well known techniques, Shlaer-Mellor, and Rumbaugh, were compared and contrasted in order to determine which would be most applicable to Rome Lab's problem. The overall results of the research provided Rome Lab with excellent information on which to base their decision of which techniques should be utilized for the future development effort. This paper provides the reader with background information from which the choice of an appropriate object-oriented analysis technique can be made. The paper concludes a very exciting and enjoyable eight weeks at Rome Laboratory.

Table of Contents

List of Figures

1.1 OBJECT ORIENTED METHODOLOGY

Object orientation can be loosely described as the software modeling and development (engineering) disciplines that make it easy to construct complex systems from individual components. The intuitive appeal of object orientation is that it provides better concepts and tools with which to model and represent the real world as clearly as possible. The advantages in programming and data modeling are many. Object oriented programming allows a more direct representation of the real world model in code. The result is that the normal radical transformation from system requirements (defined in user's terms) to system specification (defined in computer terms) is greatly reduced.

Object-oriented concepts and tools are enabling technologies that allow real-world problems to be expressed easily and naturally. Object-oriented techniques provide better methodologies to construct complex software units.

The three most fundamental aspects of the object-oriented paradigm are abstract data typing, inheritance, and object identity. Each of these concepts contributes to the software engineering and modeling properties of object-oriented languages, systems, or databases.

Abstract Data Types

Data types describe a set of objects with the same representation. There are a number of operations associated with each data type. Abstract data types extend the notion of a data type through hiding the implementation of the user-defined operations (messages) associated with the data type. Languages that support abstract data types provide constructs to directly define data operations used to manipulate occurrences of the data structures. In addition, all manipulations of instances of the data type are done exclusively through operations associated with the data type.

Inheritance

Inheritance is a powerful technique that organizes complex code bases. It allows the construction of new classes on top of existing class hierarchies. Inheritance relationships among entities in an object space can be expressed directly and naturally. Abstract data types and inheritance are used to model and organize the types or classes of objects.

Object Identity

The third powerful object-oriented concept is object identity. Identity is that property of an object that distinguishes each object from all others. With object identity, objects can contain or refer to other objects. Object Identity provides organization of the objects in the object space so that they can be easily manipulated by an object-oriented program.

Object orientation is relatively young. There is still considerable confusion and controversy over such basic concepts as objects, object-oriented languages, and object-oriented databases. Nevertheless, object-orientation offers some well established programming disciplines. These disciplines are laying the foundation of most of the software engineering activities of the 1990's.

1.2 RUMBAUGH OBJECT-ORIENTED PROGRAMMING METHODOLOGY

The methodology uses three levels of representation; high, middle, and low levels. All three levels describe the same problem at different levels of abstraction. The following discussion applies Rumbaugh's OMT (Object Modeling Technique) to relational database design. The initial logical data model is successively converted into relational tables and then into DBMS data definition commands.

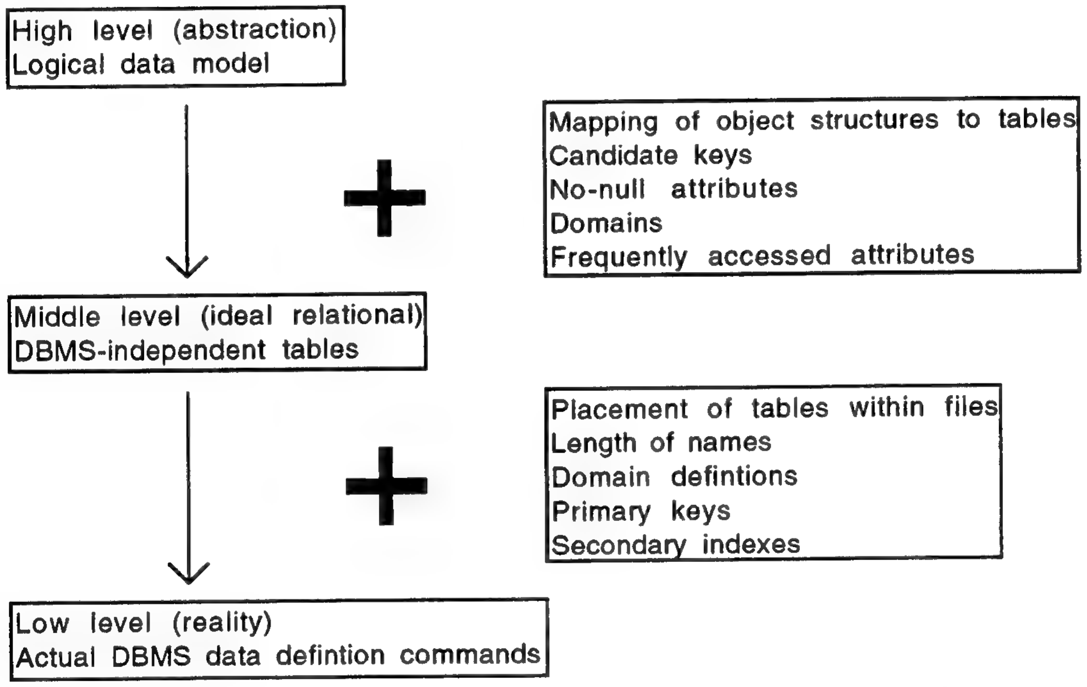


Figure 1: Rumbaugh Object-Oriented Methodology

The high level focuses on the fundamental data structure. The high level is a subset of a graphical notation that was developed by Loomis, Shah, and Rumbaugh for object-oriented programming. The middle level contains generic, DBMS independent tables. The purpose for the middle level is to decouple the general problem of mapping objects to tables, domains, and keys. The low level is the data definition language of the target DBMS. this level contains the actual DBMS commands that create tables, attributes, and indexes. The low level considers DBMS-specific details such as placement of tables within database files, a limited set of data types, and choice performance tuning mechanisms. It also deals with arbitrary restrictions such as limitations.

1.2.1 High-Level Representation

Objects

An object is a thing that exists and has identity. A group of similar objects form an object class. An object class is an instance of an object class described by attributes or fields. The boxes in figure 2 denote object classes.

Relationships

A relationship is a logical binding between objects. There are three types of relationships; generalization, aggregation, and association. Relationships are shown with a line or lines between the objects. Special symbols at the ends of a relationship line indicate how many objects of one class relate to each object of another class. It is called the multiplicity of the relationship. A small solid circle means zero or one. A straight line ending without a symbol denotes exactly one.

Generalization Relationship

A generalization is a relationship that partitions a class into mutually exclusive subclasses. Generalization may have arbitrary number of levels. The arrows in figure 2 symbolize generalization. Each box in figure 2 corresponds to an object class. Each box does not correspond to an object. The same object is being represented at each level of the generalization. Attributes are inherited from the top level down. Object modeling technique supports multiple inheritance. Each object may participate in more than one generalization hierarchy.

Aggregation Relationship

Aggregation is an assembly-component or a part of relationship. Aggregation combines low level objects into composite objects. Aggregation may be multilevel and recursive. Aggregations often exhibit existence dependency.

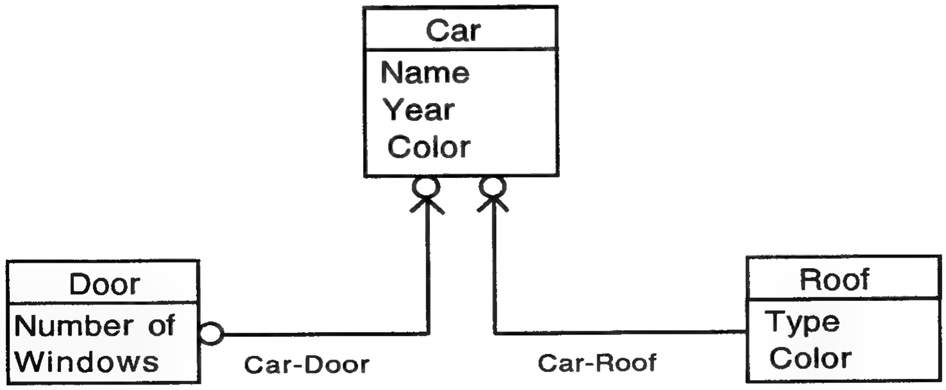


Figure 2. Aggregation Relationships

<u>Association Relationship</u>

An Association relates two or more independent objects. Associations do not exhibit existence dependency. Associations may have one or more properties.

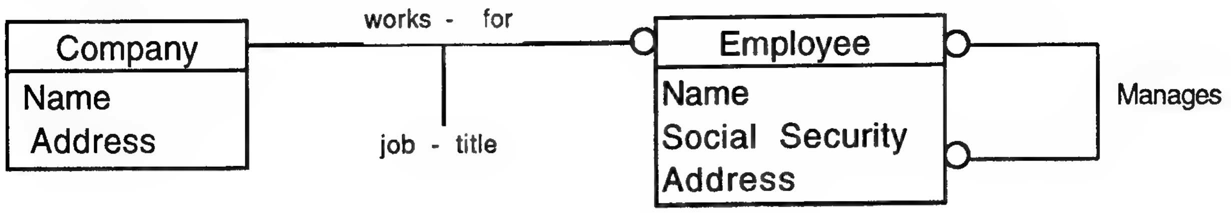


Figure 3. Association Relationships

<u>Qualification of Relationships.</u>

Qualification adds information about the end of a relationship. Qualification has two benefits; improved accuracy and more visible navigation paths. Qualification is a major advantage of the OMT approach. Qualification frequently occurs and is worthy of support.

1.2.2 Middle Level Representation

The middle level maps high-level object structures into generic tables. The middle level decouples the general problem of mapping objects to tables from the idiosyncrasies of a DBMS. In the Rumbaugh application the resulting tables tend to be in third normal form. Third normal form is an intrinsic benefit of object modeling. A table is in first normal form when each attribute value is atomic and does not contain a repeating group. A table is in the second normal form when it satisfies first normal form and each row has a unique key. A table is in third normal form when it satisfies second normal form and each non-key attribute directly depends on the primary key.

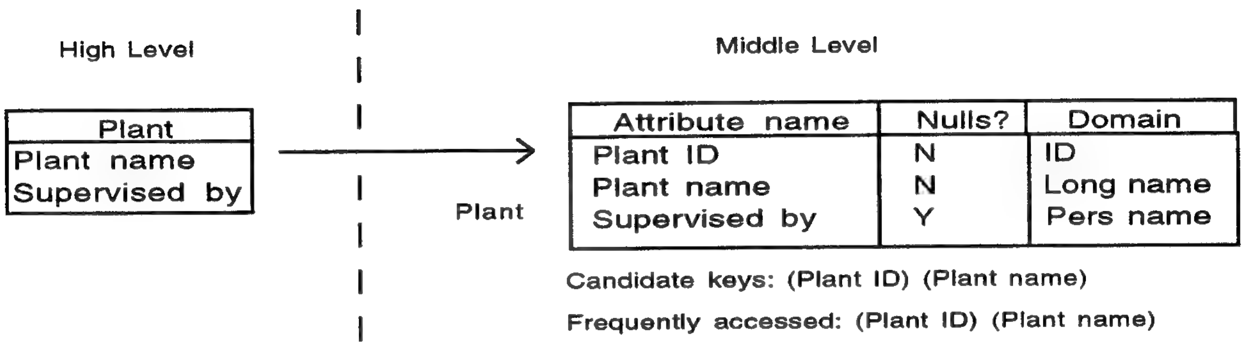
<u>Objects</u>

Each object class maps directly to one table. All object fields become attributes of tables. The Rumbaugh data modeling methodology supports the notion of object identity. Object identity is implicit in object diagrams and must be made explicit in tables.

One advantage for having a strong sense of object identity is that object IDs are immutable and completely independent of changes in data value and physical location. The stability of object IDs is particularly important for relationships since they refer to objects. Object identity provides a uniform mechanism for referencing all objects. Each attribute has a domain or set of legal attribute values.

Generalization Relationship

A generalization relationship has one superclass table and multiple subclass tables.



Attribute name	Nulls?	Domain
Plant ID	N	ID
Plant name	N	Long name
Supervised by	Y	Pers name

Candidate keys: (Plant ID) (Plant name)

Frequently accessed: (Plant ID) (Plant name)

Figure 4: Middle Level for an Object

Aggregation Relationship

Many-to-Many relationships by necessity map to distinct tables. This is a consequence of normal form. One-to-one and one-to-many relationships may be mapped to distinct tables or merged with a participating object.

Association Relationship

Properties of an association become attributes of the association table.

1.2.3 Low-Level Representation

The low-level is the data definition language of the target DBMS. This level contains the actual DBMS commands that create tables, attributes, and indexes.

Primary Keys

The primary key must be unique. None of the participating fields may be null. The middle level identifies candidate keys. One candidate key must be chosen as the primary key.

Secondary Indexes

Secondary indexes in most relational DBMS serve a dual role. They improve the performance of some queries by quickly finding the rows with a certain attribute value. Secondary indexes can also enforce the uniqueness of candidate keys. The middle level enables a clear indication of intent. The low-level generates executable code.

The Rumbaugh OMT-based approach to database design has many advantages. It is:

1. Intuitive, easy to use, and easy to understand. Non- DBMS application experts are able to read OMT diagrams after only a few hours of explanation.

2. Expressive. It provides a richer set of constructs for modeling data than alternative approaches.

3. Extensible. It accommodates changes in the scope of the data model and ports to other DBMS.

4. A useful level of abstraction. It matches real world problems well and maps naturally to a relational DBMS.

5. Good performance. It is easy to visualize patterns of access to the data when using the OMT.

6. Promotes database integrity. Object-derived tables tend to be in third normal form.

7. Improves integration. The object paradigm helps bridge the semantic gap between databases and applications.(Rumbaugh,425)

1.3 SHLAER-MELLOR OBJECT ORIENTED METHODOLOGY

The Shlaer-Mellor methodology is similar to the Rumbaugh. It uses some of the same key terms for each program.

An object is an abstraction of a set of associations that hold systematically between different kinds of things in the real world (Shlaer-Mellor, pg 47). The methodology includes attributes and the relationship.

Unconditional one-to-one Relationship

In a one-to-one relationship, a given instance of type A object is associated with one and only one instance of a type B object. Furthermore,every instance of the type B object must have an instance of the type A object so associated. This form of relationship is indicated by the symbol (1:1).

Unconditional one-to-many Relationships

Is an unconditional one-to-many relationship, denoted by the symbol (1:M), a single instance of a type A object is associated with one or more instances of a type B object. Every instance of the type B object is associated with exactly one instance of the type A object.

Unconditional many-to-many Relationships

In a many-to-many relationships, denoted by the symbol (M:M), every instance of the type A object associated with one or more instances of the type B object, and every instance of the type B object is associated with one or more instances of type A.

1.3.1 Shlaer-Mellor Software Development Process

The following sections deals with an overall software development process which makes repeated use of the information model.

The process breaks out into four phases:

1. Analysis of the Problem
2. External Specification
3. System Design
4. Implementation and Integration

The Analysis Phase: An Object-Oriented Approach

The first step is to build the information model to define the conceptual units that will be used.

Once the information model has been built, instances of same objects can be regarded as undergoing a kind of "life cycle". The life cycle concept can be formalized with the aid of a state model. Such a model can be shown in a graphical representation known as a state transition diagram. Usually a box will denote a state, which represents a stage in the life cycle for instances of the object concerned. Behavior over time is really what is being formalized by the state model. Arrows shown on the state transition diagram are transitions which take place when an event occurs. The event is written next to the transition to show what causes the transition. The final component of the diagram is the action. Actions are associated with states, so that when an entry into a state occurs all the associated actions take place.

Actions and Data Flow Diagrams.

The data flow diagram contains three components.

1- data flows, which are represented by labeled arrows;
2- processes, represented by circles; and
3- stores, shown as pairs of parallel lines.

The stores are derived directly from the objects of the information model and are named to correspond. The data flows are the active units of data-packets of data containing the named attributes. To specify exactly what a process must do, a detailed description for each process must be provided. Process descriptions can be rendered through a variety of notations and languages, including narrative natural-language text, mathematics, a pseudocode based on relational query languages, and decision trees.

Integrating the Analysis Models

The analysis models state the conceptual units of the problem (the objects), The life cycle that the instances of each conceptual unit go through, and the processing required to draw each unit through its life cycle. These aspects of the problem domain are expressed in the information model, the state model and the data flow diagram. The models are integrated, with the information model providing the foundations for the other two as follows:

- State transition diagrams are built for the objects that exhibit a life cycle or operational cycle.
- State transitions cause actions to take place. These actions are represented as processes on data flow diagram
- Each object on the information model becomes a store on the data flow diagram.
- Processes accept and produce only the data which is defined in the information model.
- Processes may create events which appear on the state transition diagram.

It is these relationships between the information, state, and process models which lead to the structure of the Analysis Phase.

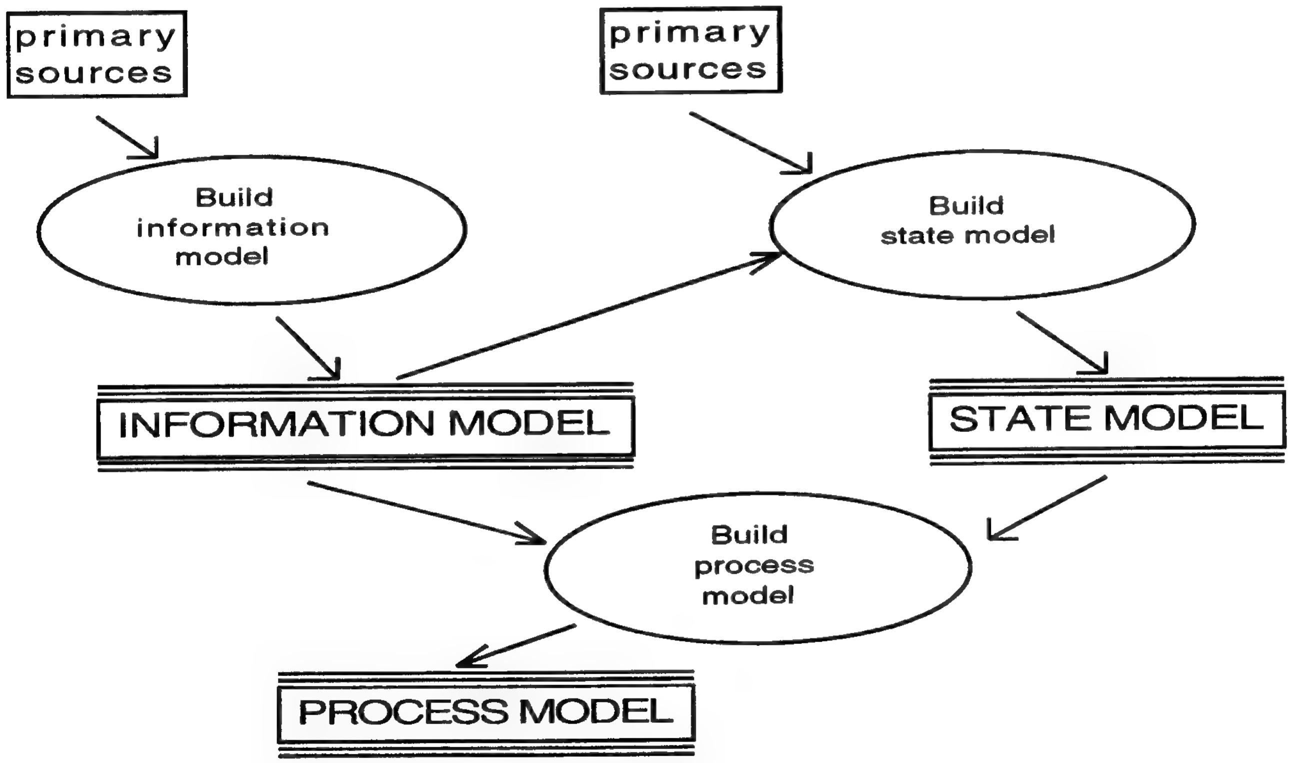


Figure 6: Summary of the Analysis Phase

The External Specification Phase

The analysis documents, taken together, provide an abstract model of a system; an information-based entity that behaves in a systematic, defined, and predictable manner to inputs or stimuli. The purpose of the external specification phase is to make decisions about what the computer should do and what it should not do.

One strategy that may be used to state the external specification is to focus on the automation boundary, the imaginary line between the portion of the abstract system to be held in computers and the portion to be left outside. Such an external specification can be based on the concept of external events; those events of the state models which cross the automation boundary. An external specification is only a statement of what a system is to do, not a statement of how the system is to accomplish its assignments. Most external specifications are open to several alternative system designs.

In the external specification phase, only two pieces of information are being added. First, there is a statement of the system boundary, and second, there is statement of how the system appears to those who interact with it. This new information is strongly supported by the analysis models and particularly the information model.

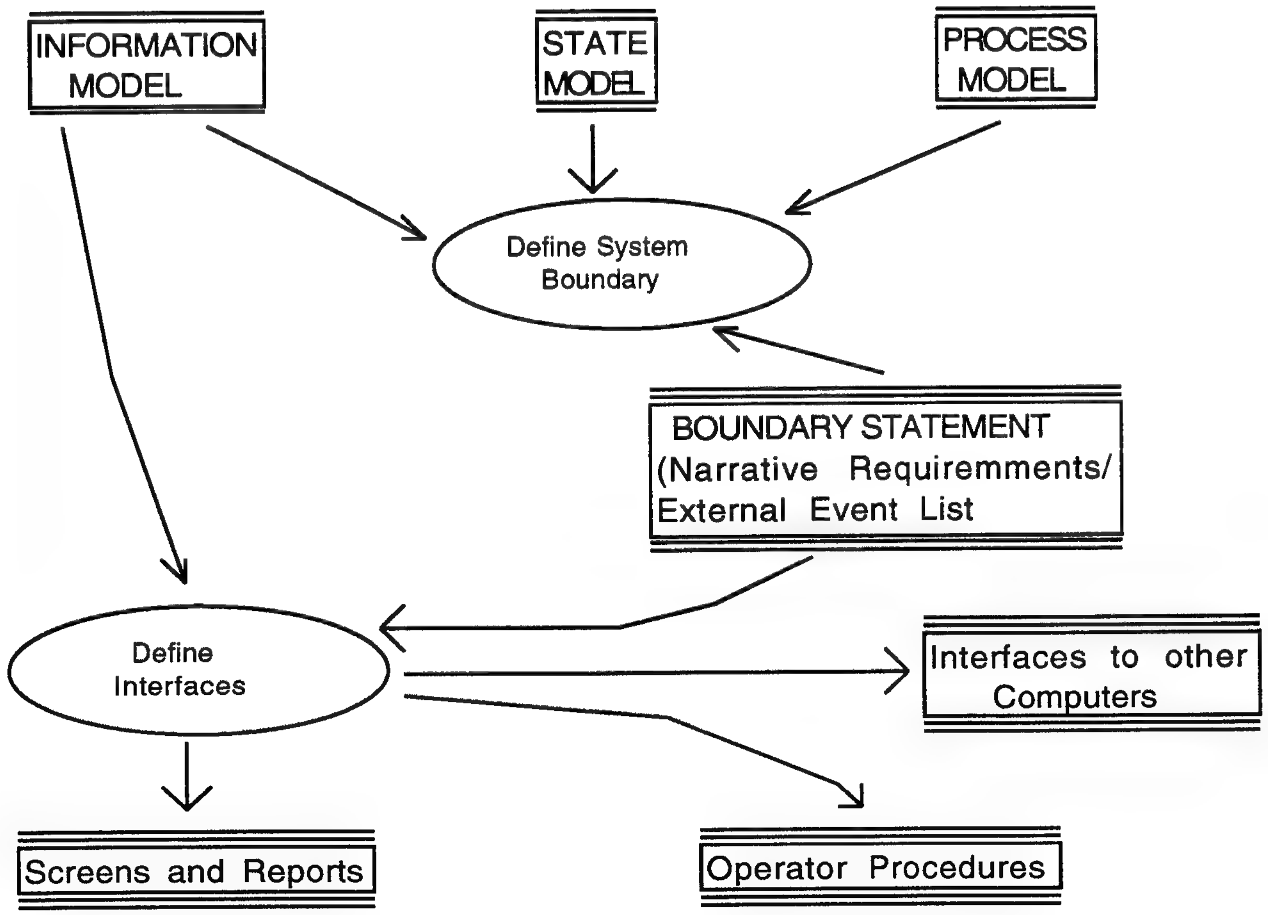


Figure 7: Summary of the External Specification Phase

The System Design Phase

At this point a complete specification of the system has been constructed, and now a system must be developed. The design work can be broken down into the following steps. Software Architecture, Design of System Information Content, Data Structure Design, and Partitioning into Programs.

In the first step, all or no work may be involved if a commercially available database system can be used. Within the second step, the decision of what information needs to be available in the automated system is done. The reduced information model can be conveniently rendered as an information structure diagram that shows only the retained objects, attributes, and relationships. The next step is to translate the conceptual view of the data into data structures used for implementation. Of major importance in many applications is the speed of access to the data.

Whereas in previous phases the information model was valuable for the application knowledge it conveyed, in the System Design Phase it plays a different role: that of conveying the intrinsic structure of the information that the system must process.

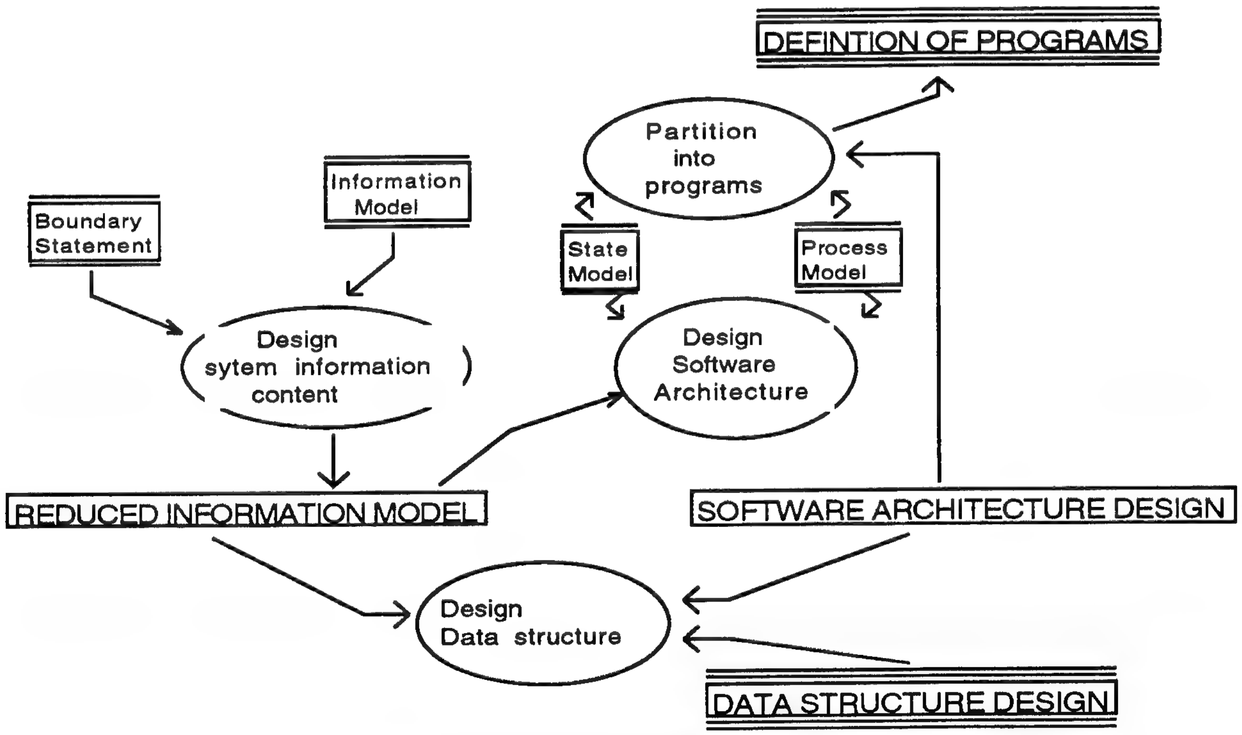


Figure 8: Summary of the System Design Phase

The Implementation Phase

The implementation phase is concerned with getting the data and programs prepared and tested. This is the final stage of the four step process. This is the actual construction of the system to the requirements already made.

Data Collection

This aspect of implementation is driven primarily from the reduced information model. The reduced information model should be looked at to see if there is any data that isn't modified during operation that is required in the automated system.

Program Design

In this step the programs are parceled out to implementation specialists for design. The

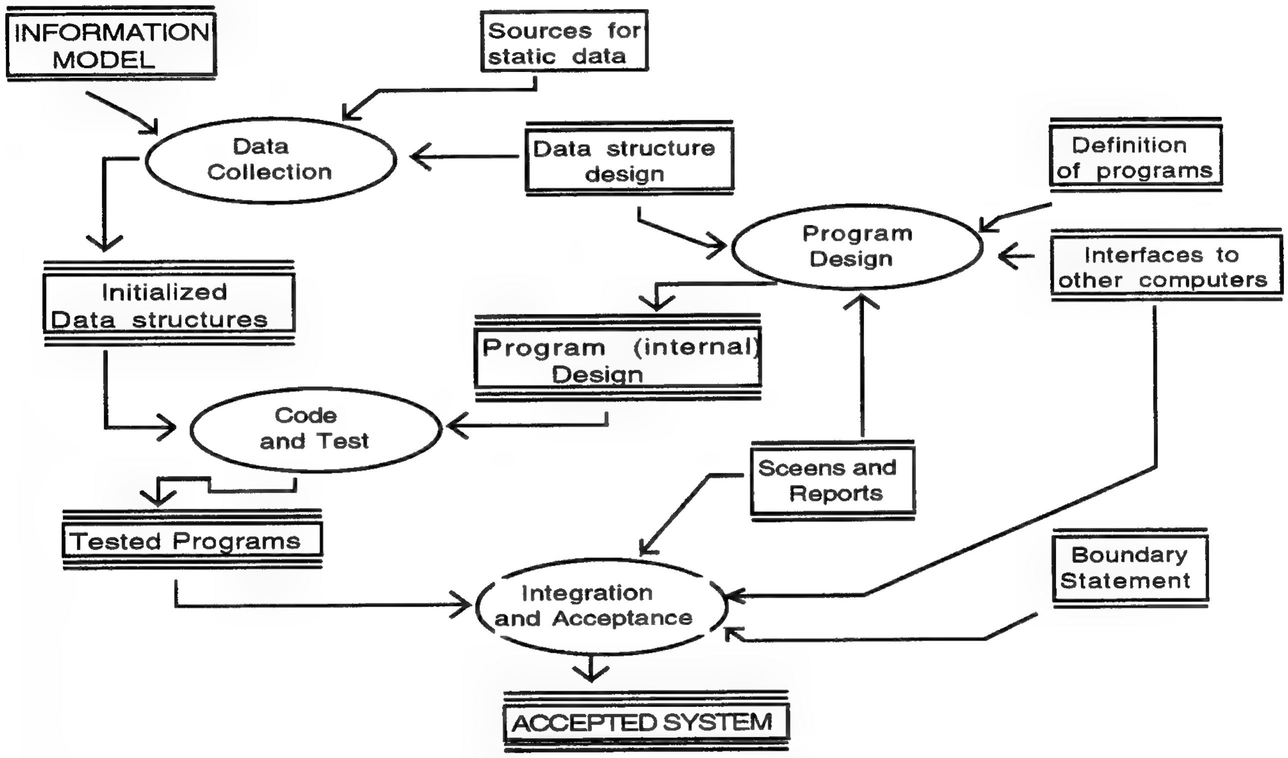


Figure 9: Summary of the Implementation Phase

primary tool applicable to design of the internal structure of a program is the structure chart.

Code and Test

This step consists of the construction of the programs themselves in accordance with the program design. Each module should be coded and tested separately, and then progressively integrated, each module with its callers or subordinates, to produce the program.

Integration and Acceptance

Integration consists of putting the various programs and data together to produce a system. Integration is the test of the system design. Integration and acceptance require good planning well in advance of their execution.

The implementation phase consists of the actual construction of the system and a check to make sure it does what it is supposed to do. With the exception of the data collection step, the information model enters into the final phase of the software development process only indirectly; through materials produced in earlier phases that were derived in part from the information model.

The development process has been divided roughly into four phases. In each step the information model has played a significant role, most p articularly in the early stages. In the later stages, the information model is applied chiefly in data structure or database design. Because the operations were derived from the life cycles of the objects, the information model has also had a strong, if less direct, impact on the code.

1.4 CONCLUSION

Both methodologies have been proven to be successful based on their performance in industry. The more popular model is the Shlaer-Mellor. In this model the key factor is in the information model, which plays a major role throughout the whole model. Both the Shlaer-Mellor and Rumbaugh model are intuitive, easy to use, and easy to understand. But the Shlaer-Mellor model remains to be the more popular throughout the industry. Its processes and steps are quicker to follow and understand to that of the Rumbaugh model. The Shlaer-Mellor technique is appropriate for complex problem environments (real-time control systems, decision support systems, or knowledge based systems), as well as for straightforward applications with minimal processing requirements. The Rumbaugh technique has been successfully applied the straightforward applications which include database problems. The main drawback to both and even to the whole process is the time consuming manual steps involved in object-oriented analysis. That has led other companies to try to create a software package that would decrease the time needed. As of now there is only one package which decreases the time in the Shlaer-Mellor model. Teamwork/OOA automates the entire Shlaer-Mellor OOA methodology, including support for recursive design and all specified work products. The Teamwork/OOA reduces system complexity by allowing analysts to graphically describe a system from many perspectives. These perspectives include Information models, Object Communication Models, State models, and Process models. The Teamwork/OOA improves productivity by automatically deriving and generating many OOA work products. Teamwork/OOA improves software quality by ensuring that all work products conform to the methodology, and that they are complete and accurate.

Object modeling has improved clarity of thought and the ability to communicate during design process. It will continue its importance and will improve as more and more people become familiar with it.

Bibliography

Abnous/Khoshafian, Object Orientation Concepts, Languages, Databases, User Interfaces; John Wiley & Sons, Inc: New York, 1990.

Kim/Loachovsky, Object-Oriented Concepts, Databases, and Applications; Addison-Welsey Publishing Company: New York, New York, 1989.

Martin, James, Principles of Object-Oriented Analysis and Design; PTR Prentice Hall: Englewood Cliffs, New Jersey, 1993.

Pinson/Wiener, An Introduction to Object-Oriented Programming and Smalltalk; Addison-Welsey Publishing Company: New York, 1988

Rumbaugh/Blaha/Premerlani, "Relational Database Design Using An Object-Oriented Methodology." Comunications of the ACM; April 1988 Volume 31, Number 4.

Shlaer/Mellor, Object-Oreinted Systems analysis; Yourdon Press: Englewood Cliffs, New Jersey, 1988.

RADAR NOISE ELIMINATION UTILIZING NEURAL NETWORKS

Eric D. Nielsen
High School Apprentice

Rome Laboratory
525 Brooks Road
RL/C3AA
Griffiss AFB, NY 13441-4505

Final Report for
High School Apprenticeship Program
Rome Laboratory

Sponsored by:
Air Force Office of Scientific Research
Griffiss Air Force Base, Rome, NY.

August 1993

RADAR NOISE ELIMINATION UTILIZING NEURAL NETWORKS

Eric D. Nielsen
High School Apprentice

Abstract

All manners of electronic devices suffer from a problem called noise. In all situations, noise results in either a loss or a gain of information. With radar this problem manifests itself in the form of pseudo-contacts -- contacts that do not exist in the real world. These noise contacts can cause numerous difficulties for people and computers that must use the radar data. One of the more common applications that uses raw radar data is a computerized tracking system. When one of these programs receives data distorted by noise, the tracking program will have problems in deciding which contacts to link. This can lead the tracking application to link a valid contact to a noise contact, a noise contact to a valid contact, or a noise contact to another noise contact. The pairing of contacts is vital to the calculation of tracking data for the tracked contacts. All three of the mentioned situations can lead to difficulties. Planes sent out to intercept a contact might not find it, wasting valuable resources. To help solve this problem, a neural network attempted to filter out the noise before it reaches the tracking stage. A C program, that takes the raw radar data and calculates other more important values from it, controls the execution of the neural network. The C program passes the neural network a list of values for each contact. The network then determines the validity of each contact. Tracking programs and other applications that need this filtered data can then receive it from the C program.

RADAR NOISE ELIMINATION UTILIZING NEURAL NETWORKS

Eric D. Nielsen

INTRODUCTION

All varieties of electronic equipment experience a phenomenon known as noise, the severity of this interference may vary, but all devices experience some form of electronic noise. With modems this manifests in the form of line noise, which can corrupt vital information. Static disrupts radio signals as a result of electronic noise or interference. Radar does not experience the loss of information that plagues the previous examples, however. Radar experiences a gain in information. Pseudo-contacts appear on the radar screen. These contacts exist only in the circuitry of the radar. Something as mundane as a weather system, such as a thunderstorm, can interfere with the radar signals causing noise. Tracking programs sometimes can not link some occurrences of noise, however, to any external occurrence. Thus the application must assume that the contact is valid. Some of these pseudo-contacts appear and disappear intermittently. Other noise signals can appear for some arbitrary amount of time and then vanish. This complete randomness of the noise signals can make the noise very hard to filter out and identify.

Computer scientists researching artificial intelligence developed neural networks in an attempt to make computers intelligent. As intelligence is most often attributing only to living animals, they modeled their system after the brain, the organ responsible for this intelligence in animals. Numerous layers and nodes compose this computer simulation of the brain. Each node roughly corresponds to a neuron in the brain. In the human brain, some neural connections may span several layers. In the computer these connections usually just span from one layer to the next, but it is possible to have it span more. Doing this, however, is uncommon. Most networks appear unhindered by this difference between the actual brain and the machine representation. No machine is intelligent, yet, but the number of nodes in most neural networks is astronomically less than the number of neurons in the human brain.

The operation of a neural network follows a set procedure. The first layer of nodes in the neural network, often called the input layer, receives the initial values, called the input values. All nodes connected to any given input node are passed through the connections, the input node's input value. Each connection has a weighted value through which it modifies this value as the input value is being relayed to the next node. Each node in the next layer then computes a value to sent to the layer that it connects to based on some function value of its collective inputs. This corresponds to the individual neurons in the brain firing after surpassing a certain threshold value.

DISCUSSION OF PROBLEM

This task of eliminating electronic noise from raw radar data is a complicated task. Splitting this into two smaller tasks makes the project more manageable. The first division is the actual neural network. The neural network will be responsible for the determination of the validity of the radar contacts. The second division is the C program that calculates values for and executes the neural network.

The first of these divisions, the neural network, in some ways is the simpler of the two. It is easy to construct a neural network, but constructing one that can solve the problem is another story. Careful thought has to go into the decision concerning the number of layers and the number of nodes in each layer of the neural network. Once created, the training of the neural network begins. During training the application executing the neural network knows the desired answers to the given input values, this knowledge can teach the network from its errors. A process call backward-propagation of errors modifies the weights and biases of the connections and nodes, respectively, when the neural network's answer is in error. This automation of this procedure saves the user time at this stage, but the job of figuring out what the desired answers can still be a daunting task. The creation of the pattern file containing the answers and responses is one of major task at this stage.

Specifying sample combinations of the input values that construe a valid contact and sample combinations that do not construe a valid contact creates this pattern file. Using actual radar data is one of the best methods, but this sample data must contain noise. Which contacts are real and which contacts are examples of noise determines the accuracy of the neural network's training. Training on a small set of sample data the neural network causes erroneous behavior. Unforeseen patterns in the data the neural network trains on causes this type of behavior. If there is little or no sample data available, then the programmer must create a pattern file. The programmer must supply what he or she believes is a good sampling of the possible input values for the network, with desired output values. If these values are incorrect, the network will not function as desired.

The C program, on the other hand, is the driving force in the running of the noise elimination system. This C program receives from the radar the distance and azimuth to a contact. At the same time the computer program also receives transpondence information about the contacts picked up by the radar. This transpondence information is one of the most helpful values in classifying contacts into two categories, actual contacts and non-existent contacts. The other data, the distance and azimuth values, are enough to draw a point on the radar screen showing the location of the contact.

The program then calculates values such as speed, acceleration, persistence, heading and change in heading. To calculate these values, the program matches contacts in the current sweep with contacts from the previous sweep. Some of these values require more information than just the simple distance-azimuth given by the radar. To calculate these values such as acceleration, persistence, and change in heading, the program analyzes data from the radar sweeps preceding the previous sweep. To accomplish this, the program stores the values for speed and heading in a structure. This allows the program to access and modify this data with ease. For example, the calculation of acceleration requires the speed values from the current and previous sweeps. The speed value for the current sweep requires the pairing of contacts from the current and previous sweeps. The results of the previous sweep's work contain the speed value for the previous sweep. The number of radar sweeps required to furnish this complex information, such as acceleration, is three and the number of sweeps needed for the simpler information, such as speed, is two.

Two special cases, or exceptions to the usual rules of contact pairing occur when a contact approaches the imaginary line drawn from the radar to a point due north of the radar. Each radar sweep starts scanning due north and the sweeps clock-wise around the circle. If a contact moves west across the imaginary line during a sweep, its image appears twice on the radar screen for that sweep. While this is not an error, it causes problems with the paring of contacts. In this case the pairing of the contact requires the use of data from the previous sweep and the current sweep, most cases only require data from the previous sweep. The tracking program must be careful, however, as this double image corrupts tracking as it increases the likelihood of links to incorrect locations occurring.

The other exception occurs when the contact moves east across the imaginary line. When this happens the radar will not detect the contact for one sweep. This causes the contact to skip a sweep. This also is not an error. It is the result of the manner in which radar operates. Still, it causes problems when paring contacts. A point from the radar sweep preceding the previous sweep now represents this contacts previous location. Solving this problem requires storing the radar data from three sweeps, one for the current and the two previous sweeps. A contact that is close to this imaginary line does not always cross it, causing even more problems.

METHODOLOGY

The first stage of this project was learning how to use PlaNet, the neural network package. It came with several demonstration networks and pattern files that were very helpful. After acquiring a familiarity with the PlaNet package, the operator conducted experiments to determine the ideal network configuration required to solve this problem.

As the example network had four input nodes, two hidden nodes (the name for a node that is not an input or output node), and four output nodes, the first experimental network was a slight variation on it. A 4-2-4 network is a simpler way of saying the network configuration. Then, three input values and a single output value define the external interface of the network. The three input values corresponded to speed, persistence, and size. Acceleration later replaced size with the discovery that the radar used for this application gives only azimuth and range data, not size or some of the other values previously used. This sample network had three input nodes, two hidden nodes, and one output node making it a 3-2-1 network. This first network, however, was not efficient at learning.

The setup of the PlaNet neural network package allows each input node to receive any of eleven values. These values range from 0-1, stepping in units of 0.1. PlaNet uses the letter "a" to represent the number one. A six, for, example represents 0.6. This means that with three input nodes any of 1331 combinations are possible. The hidden nodes of the network are preeminent nodes in the network; their configuration will either allow the network to succeed or cause it to fail. These hidden nodes have to find some way of representing a correlation between the input and output values. With only two nodes it was not possible for the network to develop the required algorithm to deal with the immense number of possibilities.

The next network was a 3-25-1 network with the increase in size it might be able to deal with the combinations better than the previous network had. With the running of the neural network, however, the truth surfaced, it still required enlargement of the neural network's size. The size then increased to a 3-100-1 network and later to, what seemed to be a behemoth, a 3-500-1 network. Both of those too still seemed to be too small. It appeared to require a network of four or more layers. A neural network composed of four layers, a 3-75-50-1, was the next attempt, but this one also failed to achieve good results. In the PlaNet package there is a value called errlimit. This variable determines when the neural network is accurate enough to use with real information. Initially set to 0.01% error, the errlimit acted as the driving force behind the search for more accurate neural networks. Most of these tests were always staying above 3.0% error, showing the orders of magnitude by which they were off.

A network consisting of five layers, 3-100-50-25-1, was next and it performed better than any of the others. Two networks with seven layers followed to see if their results were any better. The layouts of these were 3-10-20-40-20-10-1 and 3-400-130-40-13-1. Both of these two networks performed slightly better than the five layer network, but were extremely slow. This meant that the five layer network was the best choice for now.

Now the program defined transpondence as the fourth input value. Transpondence is a signal that most airborne contacts should possess. Machines use transpondence to help identify planes from a distance. The program interpreted this value as either on or off. A transpondence value of 1 means that transpondence is on. This represents that the contact answered the transpondence query. If the value is zero, representing off, then the contact did not answer the transpondence query. This now made the number of possible combinations 2662, or 2*11*11*11. Transpondence as the fourth input value required the addition of a fourth input node, making the current network a 4-100-50-25-1 network. This neural network was first trained with only the second half of the new pattern file. This half contained only patterns that had the transpondence value turned on. It managed to learn this pattern file with ease. With the addition of the other half, however, it managed to make no headway into learning the new patterns. The next time the neural network received the complete set of data. A larger neural network, a 4-200-100-50-1 network, also received this data, as the number of possible input values increased. This larger network ran about five to six times slower than the smaller network and did not seem as well equipped to handle to the problem.

The first pattern file contained only data concerning contacts that possessed transpondence or acted like missiles. When the next revision in the pattern file occurred, helicopters, fighters, and bombers became reality to the neural network as well. PlaNet then ran this larger data set through the neural network for 100 epochs, or training cycles. The other tests had all lasted either 50 epochs or 100 epochs.

While experimenting with the network controls, a mistake manifested itself. The networks trained themselves on the order that PlaNet presented the patterns. There is an option to present the patterns in either sequential or random order. The default is sequential. Invoking the random option led to an extreme increase in the error rate for a few epochs, but very soon this change allowed the networks to achieve lower error values than before. This improper training might have occurred due to the arrangement of the patterns in the pattern file. The second half of the patterns in the pattern file all had transpondence turned on. This also implies that all of these patterns represent valid contacts. As all of these patterns are at the rear of the pattern file, the program would come across these patterns and then it would reinforce any sequence of weights that said yes this is a contact, even if it did not stem from the transpondence value. This in turn led to too much reinforcement on bad techniques that hurt the system.

Networks of this size being used, five layers, took about two workdays or more to run to completion. A smaller neural network could execute more epochs per day than the larger neural networks. A reduction in the size of the neural network allows for a potentially large increase in speed. This led to the creation of a smaller, faster neural network, a 4-75-25-1 network. The older 4-100-50-25-1 contained 6675 connections between nodes that required calculations twice per pattern per epoch, once to calculate the network's response and then once to train it. The new network had only 2200 connections, about two-thirds less. The time required to execute a single epoch dropped from over two minutes down to under a minute.

The new, smaller neural network worked much better than expected. It achieved the suggested minimum error value in only 3 epochs more than the larger neural network and accomplished this in about half the time.

Knowing that in this type of application it is important to have little or no error, the default error rate of 0.01% reasonable. Wondering if lower error values were possible this value became 0.0001% error as a test. The network, however, even after 150 more epochs was unable to reach this goal.

Noting that smaller neural networks resulted in an increase in speed and accuracy, another even smaller neural network received a chance. This new neural network consisted of only three layers. With only 500 connections this network's configuration was 4-100-1. This one failed to cope with all the patterns, however. A network in the middle a 4-200-1 with 1000 connections attempted the same feat, but it had the same problems as the three layer neural network.

The discovery that the radar could not give size data and could only give the azimuth and distance values complicated matters. In the attempt to solve this problem the writing of a computer program, in the C language, to calculate the needed values began. This program's job would be to receive the data from the radar, process it and calculate the necessary values, pass these values to the neural network, fetch the neural network's responses and then pass the results to a tracking program.

Two rather difficult tasks complicated this C program. The needed information on the contacts required the use of a small scale tracking application. The creation of noise free data to pass to tracking programs is the major goal of this project. Doing this requires a tracking module that handles noise, better than the existing applications. This creation of a tracking module is no where near complete. Currently it uses a very simplistic rule, pair each suspected contact with the last reachable contact in the list. This rule filters out only extremely isolated contacts, however. Ideally this system, the combination of programming and neural networks, replaces the existing tracking systems. Now it acts as a pre-processor to the tracking system.

Second, sending data to the neural network complicated matters even more. A subshell controls the neural network. The C-shell on top, however, usually allows commands to drop straight through to the subshell, under normal use of the PlaNet package. When the C program executes, however, it executes in a different subshell that can not find the PlaNet program, even when started from within the PlaNet shell.

The solution to the second problem took the form of running both PlaNet and the C program from within a shell script. For some reason this guaranteed that both were in the same subshell and acknowledged the other's existence; when running directly from PlaNet did not work. The initial shell script called the PlaNet package to initialize it, then it loads the weights for the nodes in the network and finally it calls the C program. From within the C program, once every radar sweep, the program calls another shell-script. This shell script causes PlaNet to load a pattern file created by the C program. Then the shell script goes on to have PlaNet package evaluate each of the patterns compared to a desired answer of zero. The shell script redirects PlaNet's output to a file that the C program examines to determine which contacts succeeded in passing the neural network's test.

The problem of tracking lasted for a long time. Solutions always seemed to be around the corner. In the end, the rule added to the application was the very simplistic rule already mentioned. This is not the best rule -- it is most likely one of the worst possible. It did, however, allow the rest of the program to operate. The C application filters out some noise contacts. Currently it eliminates any contact not paired to a previously suspected contact. The program currently allows a contact from the previous sweep to act as the previous contact for more than one current contact. This has good and bad features. On the beneficial side, this allows instant tracking of a missile launched from a plane, as pairing with the plane's previous position would occur. On the malevolent side, pairing of any noise contacts near another contact's previous position also occurs. This shows the drawback of this approach. Now, hopefully, most noise contact structures' hold values that will allow elimination of them by the neural network. Otherwise noise contact will stay and will confuse the tracking module on subsequent sweeps as well.

The addition of the heading and change in (delta) heading occurred while progress was slow on the tracking module. This addition required the pattern files change as well to reflect this new data. The number of combinations rose from 2662 to close to 30000 or 2*11*11*11*11. The four-layer network previously used with much success failed to handle this increase in size. A larger four-layer network also fell short of expectations. A five layer network is currently under testing and looks promising. The five input values will allow the neural network to do a better job at classifying targets. Hopefully, this will overcome the shortcomings of the simplistic tracking rule used in the C program.

RESULTS

The large number of different neural networks tried show that this project requires people with more knowledge of neural networks, for the successful completion of this project. The amount of time required for the processing of each sweep will require that this application executes an every radar that feeds information to the tracking application. While sending all the data to one system offers the potential for better accuracy in classifying contacts, the larger amount of information prohibits this option. The neural network handles roughly 400 contacts every six seconds. The radar sweeps occur every 12 seconds. Possibly up to eight or nine seconds can be allocated to the neural network, depending on the amount of time required by the actual tracking application.

CONCLUSION

This project required more work that imagined. People with more experience in the fields of neural networks and tracking need to aid in the development of this application The neural network as it stands now, works, but is most likely not anywhere the most efficient configuration possible. The internal tracking module of the C program also requires much more work. Use of some of the data supplied in the data structures allows for the development of a much better system, given more time and resources.

Suggested ideas include making use of the 'used' field that is currently unimplemented. The suggested usage of this field is as a marker to determine whether the tracking system used that contact as a previous contact. This would allow the system the possible use of each contact a maximum of two or three times, instead of the current unlimited usage allowed. Allowing only one use, causes the tracking application to ignore missiles or other items launched from planes for one sweep, which is not acceptable. Also only a single pairing per contact, would allow tight formations to throw the tracking system into chaos when they split up.

Also the exceptions to the tracking rules currently lack implementation as well. Blank spots in the C program await this addition, but are currently turned off by comments. This means that the current system will not be able to track effectively incoming contact from due north of the radar.

This approach to the problem of radar noise elimination appears valid and further pursuit of it is justifiable. The actual degree of benefits from this sort of approach is still unknown. It potentially promises improvement over existing systems, in most situations.

References

Gregory, Scott. Technical Discussions, July 22, 1993 - August 20, 1993.

Humiston, Todd. Technical Discussions, July 23, 1993 - August 20, 1993.

Kochan, Stephen G. and Wood, Patrick H. UNIX: Shell Programming. Hayden Books, Indiana, 1990.

Miyata, Yoshiro. "A User's Guide to PlaNet Version 5.6: A Tool for Constructing, Running and Looking into a PDP Network", Yoshiro Miyata, 1991.

Miyata, Yoshiro. "PlaNet Reference Manual", Yoshiro Miyata, 1993.

Miyata, Yoshiro. E-Mail Correspondence, August 6, 1993 - August 9, 1993.

Multi-Media

Michael J. Panara
Summer Apprentice
Rome Laboratory

Griffiss Air Force Base
Rome, New York

Final Report for:
Summer Apprenticeship Program
Rome Laboratory

Sponsered by:
RDL
Culver City, CA

and

Griffiss Air Force Base

June-August 1993

Multi-Media

Michael J. Panara
Summer Apprentice
Rome Laboratory
Griffiss Air Force Base

Abstract

The importance multi-media was studied. To show the importance of multi-media a tutorial for the MacroMind Director was made using the MacroMind Director application. The tutorial showed the basics of creating a multi-media project or presentation using this application. All available media was used in attempt to show the full power of the Director as well as the full power of multi-media. This showed how easily an effective interactive multi-media presentation could be made once the basic skills are learned.

Multi-Media

Michael J. Panara

Introduction

Multi-media is the combination of many different types of media into one interactive presentation or project. It combines sound, graphics, animation, text, video, and voice. It is also interactive so the user can choose the path and rate the presentation takes.

In business and in teaching multi-media is becoming more and more popular because of its many different and important uses. It can be an effective teaching tool in the form of tutorials. These drastically cut the amount of time needed to learn how to use a new program. They also increase the amount learned about the new program.

In business a multi-media presentation can be used to display a new product or development. With the touch of a button a consumer can see an animated picture of the product or information. It can also be used in business meetings to make charts and graphs animated.

Objective

The objective of this project was to show why multi-media is so important in business and in teaching, and to show why someone should choose to use a multi-media approach to a project or presentation.

To show why multi-media is so important I created a tutorial for the MacroMind Director using the MacroMind Director application. It showed how to use all the Director's capabilities while actually using them in the presentation. It also attempted to show what an interactive multi-media production can do. Finally, it proved that an effective multi-media production can be made rather quickly once the basic skills are learned.

Background

During the project different types of equipment were used, the most important of these being the Macintosh computer. This was the system used to run the application. The system was hooked up to a standard full-color monitor. The tutorial would have had a better effect had it been hooked up to a television set. This would have made it look more like a movie.

The MacroMind Director was the only application needed to create the multi-media presentation. However, because of memory limitations many different individual movies had to be pieced together, still using the Director.

The Director allows for sound and music to be played during the presentation. However, it does not allow for sounds to be created; all sounds and music must be imported from another source. No music scores could be found to last the duration of the entire presentation. Therefore, many musical scores were needed for each individual movie. This happens also because of the memory limitations of the application.

Approach

To learn how to use the multi-media software a tutorial for the MacroMind Director was created using the Director application. This allowed me to teach about the Director while learning more about it at the same time. Because of the tools available and because of the use of "Lingo" the presentation became easier to make as I became more familiar with these commands. Also as I learned more about the application the time needed to create each individual movie was decreased.

Many options are available while using the Director application. Among these are the paint and stage options. Paint allows the user to create new graphics or modify imported graphics. When the object is created it is placed in the cast. Once the cast is either created or modified it is put into the movie on the stage. All animation and movement is created using the stage option.

To fully understand the Director tutorial one must understand how the tutorial was created. The Director has many different windows to help in the creation of each individual movie. The first being the score. The score shows which cast member is in use in a particular frame. It also shows the movement of a particular graphic as it moves across the screen.

Within the score window is the script window. This is to allow the creator of the movie to attach Lingo to a certain frame. Lingo is the language that the Director understands. It is almost like regular everyday English. With Lingo one can pause the movie, delay the movie, change the background color, or various other things

Another important window is the Cast. The Cast shows all graphics, sounds, buttons, or anything else that is in the movie. Each cast member is given a different number. (ex. A11, B12 etc.). The numbers are used to identify the cast members in the score or when they are being used in the Script. When they are used in the score each cast member will have a separate frame in the score window.

Graphic cast members can either be created using the paint option or they can be imported from another source. The graphic cast members that are imported from other sources can be altered using the Director's paint option. However, sounds must be imported from another source since the Director application does not have an option that allows for the creation of sound files. These files cannot be alter using the Director's paint option.

Adding sounds to the presentation is easily accomplished. First choose the sound row from the score. Then choose sound from the menu bar. Choose cast and then the sound or music to be played. There are two rows for sound so that two sounds can be played at once. This is a new feature of the program which makes presentations seem more interesting. Now while background music is playing sound effects can also be added at the same time

The Director also has many options to make the presentation seem more interesting. One of these is transitions. Transitions allow for the picture currently on the screen to fade out and for a new picture to fade in. They have their own row in the score. To choose, one must double click in the frame they wish to have the transition. Then a box will appear with approximately fifty choices. Choose the transition and then click on the "set transition" button.

An important element of a multi-media presentation is animation. Animation is created by using slightly different pictures in rapid sequence to give the illusion if motion. Objects that are just going to move across the screen do not need to be drawn in each individual frame. In order to use animation simply choose the starting and ending point on the stage and in the score then choose "In-between" from the score menu, and the object will automatically become animated and move across the screen. Pictures with movements in them, such as a man running, will, however, have to be drawn in each individual stage, and be put in each individual frame in the score. The Director can not automatically animate graphics in this way.

Because of the memory limitations of the Director the entire presentation could not be made using only one movie. Different individual movies needed to be connected. This can be done using the Director application. To do this first choose the script window from the score window. Then type "go to frame 1 of *movie x*" (movie x is the name of the next movie) When the end of the first movie is reached the Director will continue on into the start of the next movie with no interruption.

Results

As the project went along the MacroMind Director became easier to use. The additions in the latest version the MacroMind Director application made the presentation move along much more smoothly. The most important new skill learned was the use of Lingo. This saved much time, space, and memory for the presentation. Instead of dragging a cast member for as many frames as it is to appear, a command of "delay *x* " could be used. (x represents the amount of ticks one wishes the presentation to delay.)

Also an interactive presentation can be created using the Lingo. This is done by simply typing "when mouseDown then go to *x* " (x is the new frame number; mouseDown means when the mouse button is pressed). This adds to the project because now the user can choose the rate of the presentation. The user can now either continue ahead or go back and see something again to make sure he is clear on it.

Because of the use of these two features, as well as many others, the presentation came out looking very professional. It was of a much better quality than I thought could be attained using the Director application. I felt that it was much better than the presentation created last year.

Conclusion

A multi-media application such as the MacroMind Director is very important in business and education. Using the Director application, multi-media presentations can easily be made. It is well worth the time needed to create a multi-media presentation. The tutorials that can be created using a multi-media application are much more effective in teaching than many text books or other tutorials.

The tutorial that was created showed how to use the Director application while using the Director application. I attempted to show how to use every function of the application, while actually using it. This added to the quality of the presentation because it showed the actual application at work. The actual application is available for anyone to use. It will take some time to learn some of the more complex features of the application, but once it is learned it becomes relatively simple for even the most inexperienced users.

The presentation could have been enhanced by the use of other forms of media that were not available at this time. These included a CD player. This could have been used for higher quality sounds that would last the entire length of one movie or even the entire presentation. Also unavailable was the use of video. Because of the great memory demands of an entire video sequence this option was not used at this time.

A new feature of the updated Director application that was not used is the ability to import entire or partial Quick-Time movies. This feature allows the creator of the presentation to add a Quick-Time movie into a Director movie. The Director also allows the user to export a Director movie as a Quick-Time movie. This is done by simply choosing export from the file menu, and then choosing Quick-Time movie. Exporting the file as Quick-Time allows the movie to be transferred to another computer which may not have the Director application, but can run Quick-Time movies.

Multi-media is definitely needed in today's business and education worlds. Based on what I discovered during the creation of the project, I would say that multi-media will become more and more widely used in the next decade. It provides an easy and effective way to create tutorials and other teaching aids. With a little creativity brilliant presentations can be made using multi-media.

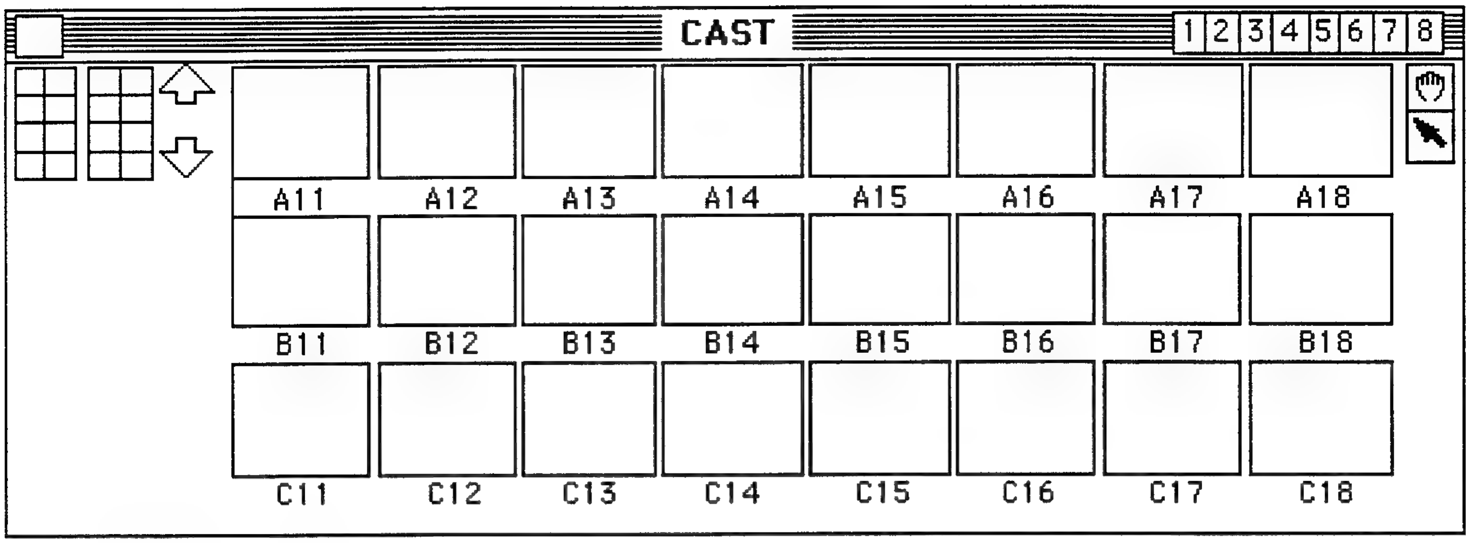

CAST
1 2 3 4 5 6 7 8
A11 A12 A13 A14 A15 A16 A17 A18
B11 B12 B13 B14 B15 B16 B17 B18
C11 C12 C13 C14 C15 C16 C17 C18

Figure 1: Cast Window

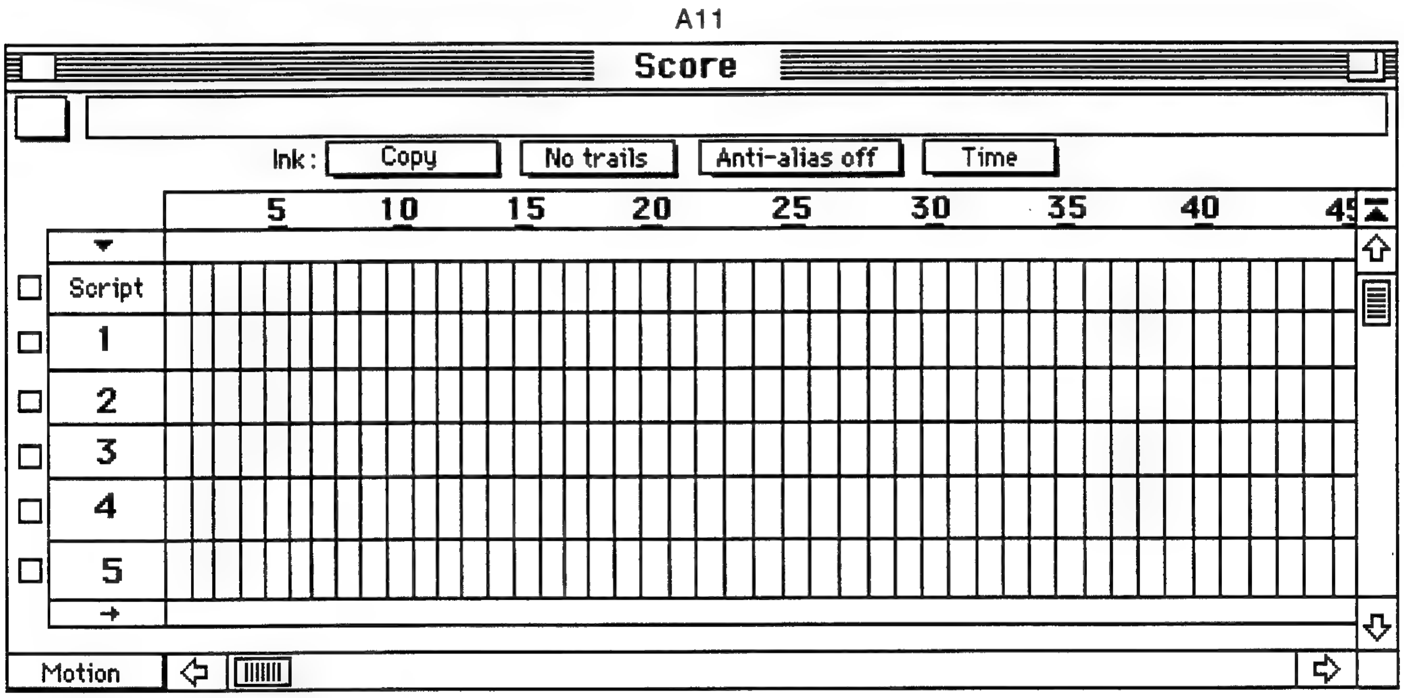


Figure 2: Score Window

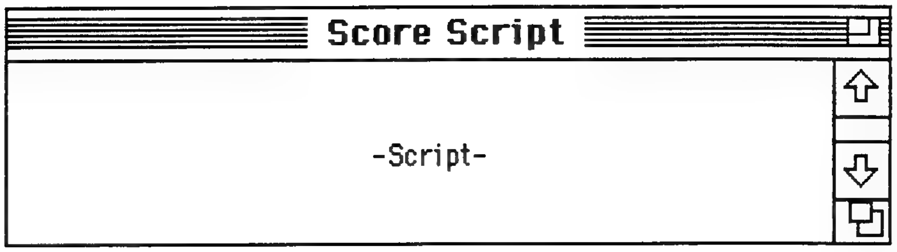


Figure 3: Script Window

A13

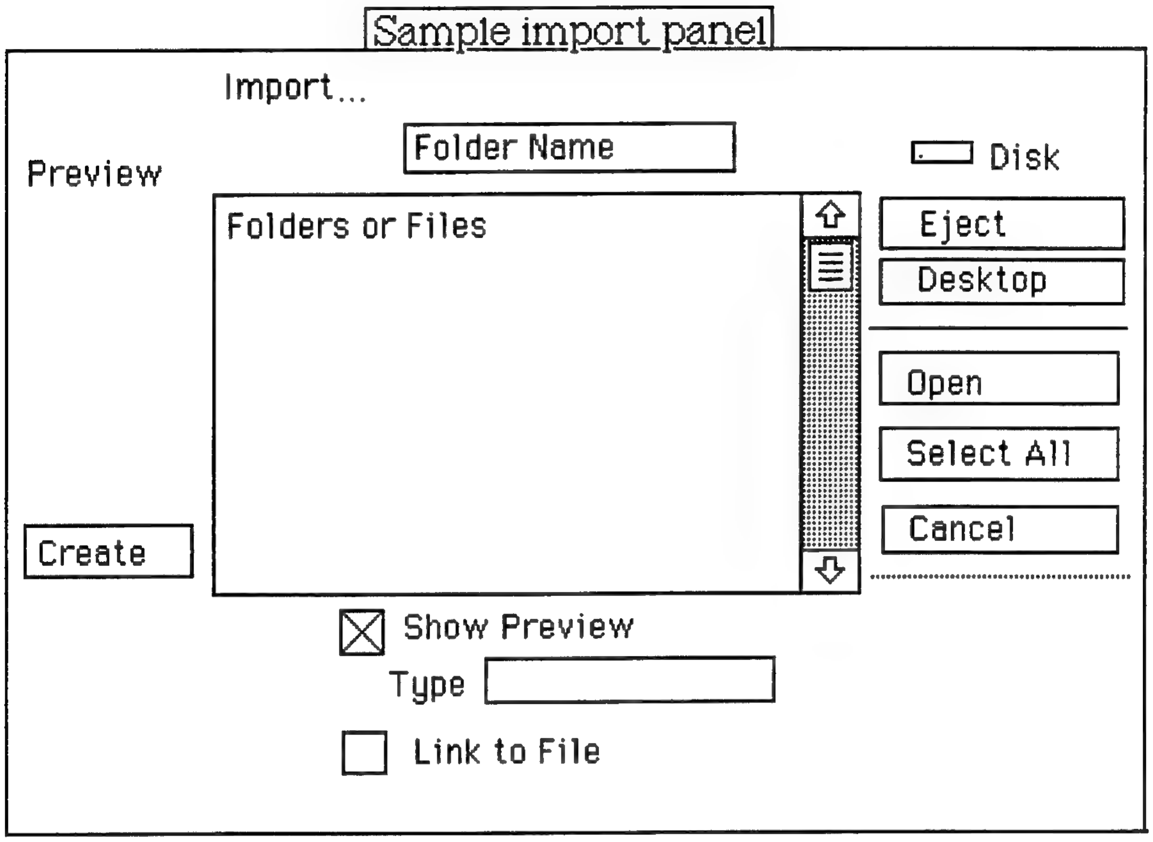


Figure 4: Sample Import panel

Study Of C Programming

Anne E. Pletl
Engineering Assistant For Software
Exercises and Verification
Griffiss Air Force Base (Rome Labs)

Rome Laboratory
Optical Processing and Communications Branch
525 Brooks Road
GAFB NY 13441-4505

Final Report For:
Summer Research Program
Rome Laboratory

Sponsored By:
Air Force Office Of Scientific Research
Bolling Air Force Base, Washington, D.C.

August 1993

A Study of C Programming

Anne E. Pletl
Engineering Assistant For Software
Exercises and Verification
Rome Laboratory
Optical Processing And Communications
Griffiss Air Force Base

ABSTRACT

C programming was the topic researched. To understand this language some basic knowledge of computers and terminology is required. In this reports I will familiarize you with the UNIX operating system and the needed vocabulary to get through a C program. The main thing about any computer language is that unless the proper commands are summited in the right order, the program with not run properly. From defining your variables to ending a statement, the program has to be one hundred percent correct or else it won't run as expected. In the following report I will show how this some what simple programming language operates and it's attributes as well as its weaknesses.

A STUDY OF C PROGRAMMING

Anne E. Pletl

INTRODUCTION

The topic studied was programming for manipulation of numerical and character data using the C language. Specific tasks included the development of programs for sorting of numeric and alpha numeric data, numerical calculations, matrix manipulation and operations for addition, subtraction, multiplication,and division. The program listings for SORT.C, INSERT.C, etc. are provided in Appendix A.

C is not considered a "very high level" language nor a "big" one because it is not specialized to any particular area of application. Its generality is what makes it more adaptable compared to more powerful languages such as Pascal, FORTRAN, and BASIC. C was originally designed for and implemented on the UNIX operating system. However, it is easy to write programs on any machine that supports C. Through this efficient language the programmer is able to create programs to run successfully.

THE UNIX OPERATING SYSTEMS

UNIX is a fairly new operating system. It was licensed to universities in 1974 "for educational purposes" and a few years later became available for commercial use. What makes this operating system so successful is that it's portable because it is written in C. The strong commercial advantage is that it runs on a range of computers from microprocessors to the largest mainframes. Its adaptability to particular requirements is due to the fact that it is written in a high-level language. The UNIX programming environment is unusually rich and productive.

C BASIC DATA TYPES

There are three basic data types in C, with different variations: integers, characters, and reals. The characters (char) are for when the programmer wishes to incorporate words (text) into the application. Reals (float or double) and integers (int) are used with numerical values. Each numerical type comes in different sizes, allowing you to select the range of the values appropriate for your needs. In the following paragraphs I will discuss further these three data types and their variations.

Integers are whole numbers with a range of values. It is undetermined as to how big an integer can be however, its range is typically 16, 32, or 36 bits. To save storage space and provide some control C has tree classes of integer storage: short (16 bits), int (32 bits), and long (36 bits). These data types, or integral types, can be read or printed using the scanf and printf commands with the %d formatting string. Integers can either be signed or unsigned. A signed integer only takes up one bit for the sign of the number whereas and unsigned integer uses all the bits for the magnitude and is always nonnegative.

Reals, or floating point numbers, are fractions and exponents. These are normally written with a decimal point or in "e" notation (see figure 1). They are defined using the float declaration and have about 7 significant digits. Doubles are used for larger significant digits that have about 64 bits per value, with the additional bits going to the fraction. Doubles are typically 14 significant digits. These are both read with scanf using the %f formatting string (floats) and %lf (doubles). Both types can be printed with printf using the %f string.

Figure 1

floating point	*scientific notation*	*"e" notation*
12.45	*1.245 x 10 to the 1*	*1.245e1*
-176.2	*-1.762 x 10 to the 2*	*-1.762e2*
0.0184	*1.840 x 10 to the -2*	*1.840e-2*

Characters allow the programmer to use numbers and characters as well. Characters encompass more than just letters of the alphabet contrary to what the popular census. They allow us to do anything within the machines character set.

Together these three make up the basic data types of the C programming language. This is just a small portion language however, it is the basics that are needed to know to begin any programming escapade.

STATEMENTS

C contains a vast array of statements, here I will review the most commonly used ones. C's simplest statement is the expression statement, an expression followed by a semicolon. It can be used anywhere the language syntax requires it. The semicolon ends the statement rather than just separating them like that in Pascal.

One such statement is the compound statement (see figure 2). Changing a compound statement to a single line statement will make the program more compact but not necessarily easier to read. These statements should be left alone unless the lines saved make the function fit one sheet of paper making it more readable. The compound statement can also be used anywhere just like the expression statement but it is not followed by a semicolon.

figure 2 compound statement

```
          {
           lineno = 0;
           pageno = 1;
           lastch = '\n';
          }
      with
       lineno = 0, pageno = 1, lastch = '\n';
```

Another type is the simple decision statement (see figure 3). These if...else statements are grouped together and a semicolon is found only at the end of each grouped statement.

figure 3 : decision statement

```
if (expression)
     true - statement
else
     false - statement
```

There is a similar statement to if...else. It is called the constant multiway decisions (switch) statement (see figure 4). This occurs in programs that process single-letter commands. The switch statement is often a more convenient, more efficient, and more readable way to make such decisions as compared to if...else statements.

figure 4 : constant multiway decision statements

```
          switch (expression)
     {
          case first-constant expression:
               statement list
          case second-constant expression:
               statement list
           case final-constant expression:
               statement list
          default:
               statement list

     }
```

There are two simple looping mechanisms in C , the do-while and while statements (see figure 5 and 6). When while is executed, a cycle of (loop) evaluating expressions and executing statements are entered. The cycle ends with a zero. Unlike the while, the do-while tests the cycle afterward. Statements are always executed at least once in a do-while loop.

figure 5 : do-while loop

```
do
   statement
while (expression)
```

figure 6 : while loop

while (expression)
statement

One advantage of C is that its statements can be very versatile.

OPERATORS

There are several basic operators when working in C. Many operators require the operand to be of a certain type. It is the programmers responsibility to make sure the operand is of the proper type so that the result will not produce overflow or underflow.

The most commonly used are the arithmetic operators (see figure 7). These all operate with a floating point, integer, and character operand. Arithmetic operators allow the program to compute mathematical equations and formulas.

figure 7 : arithmatic operators

(+) addition
(-) subtraction
(*) multiplication
(/) division
(%) remaindering

Another common operator is the assignment operators which is evaluated from left to right. It is used to assign the results of an expression to a variable (see figure 8). In short, it returns the assigned value.

figure 8 : assignment operators

```
a = (b = (c = INITIAL));
```

The logical operator takes any numerical formula and determines if it's false or true (represented by one if true and zero if false) (see figure 9 and 10).

manual. The ones discussed were used in the programs I developed.

CONCLUSIONS

In conclusion, C programming is a relatively simple programming language when compared with Pascal or BASIC. It's ability to adapt to all kinds of situations makes it the logical choice for programmers. C uses various applications incorporated into the program itself. Those in the report above were the few that I used in my research.

As you can see the possibilities for this language is limitless. That's what makes it so practical. It allows the programmer some versatility which in turn makes it the ideal language for the future.

Appendix A

```
/*
 *  Simple calculator program.
 */
#include <stdio.h>

#define MAXLINE 256

main()
{
int         inpres;      /* return value when reading in value */
char        operator[2]; /* read in operator: op followed by null*/
double      operand,     /* read in operand */
            result;       /* result of computation */

result = 0.0;
printf("Imput operator followed by operand\n");
printf("To exit type quit\n");
while (inpres = scanf("%1s%lf", operator, &operand), inpres == 2)
     switch(operator[0])
     {
          case '+': result += operand;
                    break;
          case '-': result -= operand;
                    break;
          case '*': result *= operand;
                    break;
          case '/': if (operand != 0)
                         result /= operand;
                    else
                         printf("Division by zero, ignored.\n");
                    break;
          default:  printf("Unknown operator: %c\n",operator[0]);
                    break;
     }
if (inpres == EOF)
     printf("%f\n", result);
}
```

References

1.) The ABC's of Turbo C, Hergert, Douglas, (c) 1989 Sybex, California.

2.) Algorithms in C, Sedgewick, Robert Vol. 1 & 2, (c) 1983/1988 Addison-Wesley Publishing Company, Massachusetts.

3.) Algorithms in C++, Sedgewick, Robert, (c) 1992 Addison-Wesley Publishing Company, Massachusetts.

4.) The C++ Answer Book, Hansen, Tony L., (c) 1990 Addison-Wesley Publishing Company, Massachusetts.

5.) C : The Complete Reference, Schildt, Herbert, (c) 1987 Osborne McGraw-Hill, California.

6.) C Programmig Language : An Applied Perspective, Miller, Lawrence & Quilici, Alex, (c) 1987 John Wiley and Sons, New York.

7.) The C Programming Language, Kernighan, Brian C., & Ritchie, Dennis M., (c) 1978 Prentice-Hall Incorperated, New Jersey.

8.) Learning C with Fractals, Stevens, Roger T., (c) 1993 Acedemic Press Incorperated Harcourt Brace Jovanovich, Publishers, New York.

9.) SunOS Reference Manual, Vol 1 & 2 (c) 1990 Sun Microsystems Incorperated.

10.) Turbo C Programming, Plantz, Alan C., & Brown, Willam M., & Yester, Michael, & Atkinson Lee, (c) Que Corporation, Indiana.

11.) The Unix Programming Environment, Kerninghan, Brian W., & Pike, Rob, (c) 1984 Prentice-Hall Incorporated, New Jersey.

12.) The Waite Group's Turbo C Bible, Barkakati, Nabajyoti, (c) 1989 Howard W. Sams and Company, Indiana.

Jonathan Bakert's report not available at time of publication.

INFRARED MEASUREMENTS OF ELECTROMAGNETIC FIELDS

Michael P. Decker

Rome Laboratory

RL/ERPT
Griffiss AFB, NY 13441-5700

Final Report for:
AFOSR Summer Research Program
Rome Laboratory

Sponsored by:
Air Force Office of Scientific Research
Bolling Air Force Base, Washington D.C.

August 1993

INFRARED MEASUREMENTS OF ELECTROMAGNETIC FIELDS

Michael P. Decker

Abstract

This research experiment developed a procedure for relating the temperature distribution on a RF detector to the incident power density. An infrared (IR) measurement technique was used to detect electromagnetic (EM) fields over a two dimensional area. A horn antenna was used to radiate RF energy on a RF detector material. The detector absorbed a small amount of the incident RF energy, causing the surface temperature of the detector to rise above the ambient temperature. The temperature distribution on the detector was measured with an IR measurement system.

INFRARED MEASUREMENTS OF ELECTROMAGNETIC FIELDS

Michael P. Decker

Introduction

This research experiment developed a procedure for relating the temperature distribution of the detector to the incident power density. An infrared (IR) measurement technique was used to measure electromagnetic (EM) fields over a two dimensional area. A horn antenna was used to radiate RF energy on a RF detector material. The detector absorbed a small amount of the incident RF energy. This caused the surface temperature of the detector to rise above the ambient temperature. The temperature distribution on the detector was measured with an IR measurement system.

Measuring electromagnetic fields through infrared detection is based on the Joule heating that takes place when electromagnetic energy passes through a lossy material. The energy absorbed by the detector material is converted into conducted and convected heat energy and re-radiated electromagnetic energy which is concentrated in the infrared band. The energy absorbed by the detector causes the surface temperature to rise. When the surface temperature rises above the ambient, the temperature distribution can be detected by an infrared measurement system [1]. The infrared measurement system used was a Thermovision 900 series manufactured by Agema. This is a dual band system, the long wave scanner is designed for 8-12μm of the infrared spectrum, and the short wave scanner is designed for 2-5.6μm [2]. The scanner used in the experiment was the long wave. All the equipment used is listed in Appendix A This technique has advantages over conventional electromagnetic probe measurement techniques in that it is minimally perturbing, and can be used as a diagnostic tool to map the electromagnetic fields. The temperature distribution of the RF detector was measured, over the frequency range of 2-8 GHz for separate distances between the horn antenna and the detector of 25-100 cm.

RF Measurements

The measurements were performed in an anechoic chamber with the RF detector placed in front of a horn antenna. The detector used was a carbon loaded Kapton polyimide film made by Dupont. The detector was used to absorb a small amount of the RF energy and causes an increase in the temperature above the ambient. The separation of the RF detector and the horn antenna varied from .25 to 1 meter. The IR camera was positioned at a distance of approximately 1 meter and at a 45^{0} angle to the RF detector. Figure 1 shows the test setup.

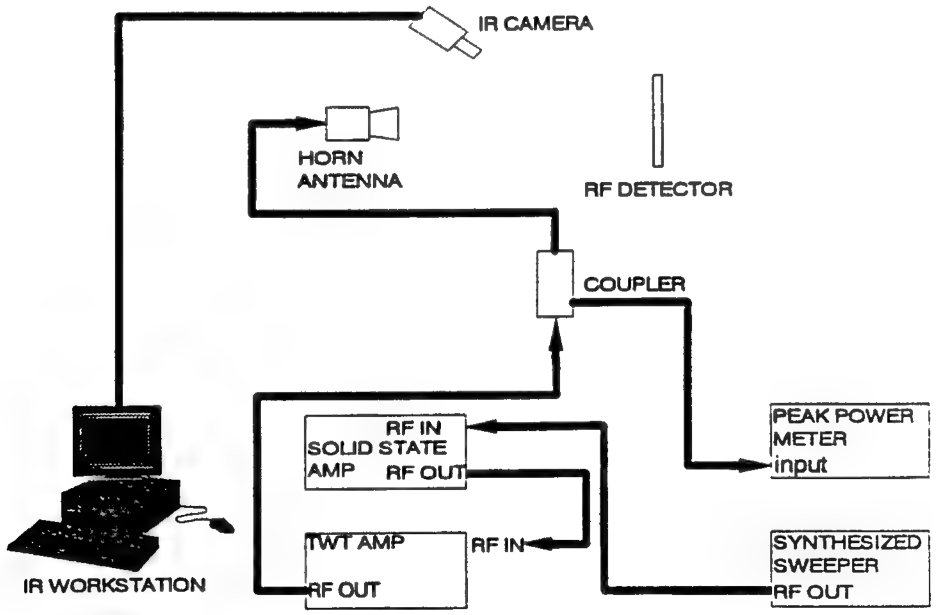
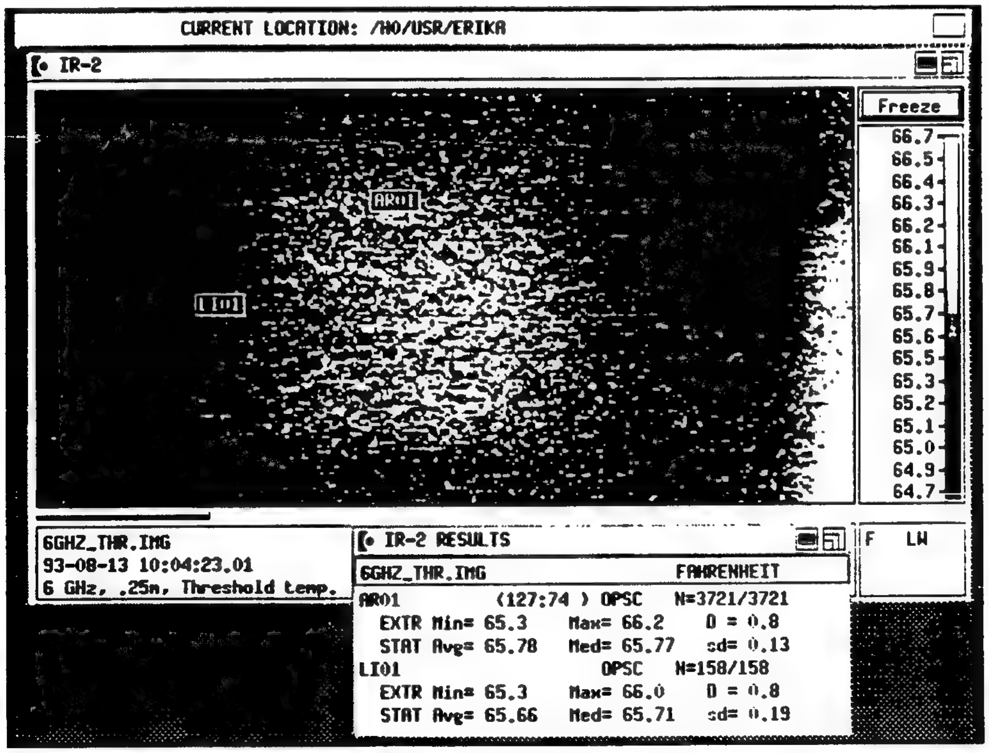


Figure 1 TEST SETUP

Seven different transmit frequencies (2, 2.5, 3, 3.5, 4, 6, 8 GHz) were used during the experiment. The ambient temperature was recorded before each measurement without an incident field present. Figure 2 shows a thermograph of the background temperature, for a detector with a resistivity of 100 ohms/sq and a separation distance of 25 cm. After the desired frequency was chosen the power level was set at -8 dBm on the peak power meter. The power level was then increased until the threshold temperature was reached. The threshold temperature is defined as an increase of one degree above the ambient temperature. After the threshold temperature and power level were recorded the transmit power was increased by one dB. The dwell time of the RF on the detector was approximately one minute, the maximum temperature for that particular power level was then recorded.



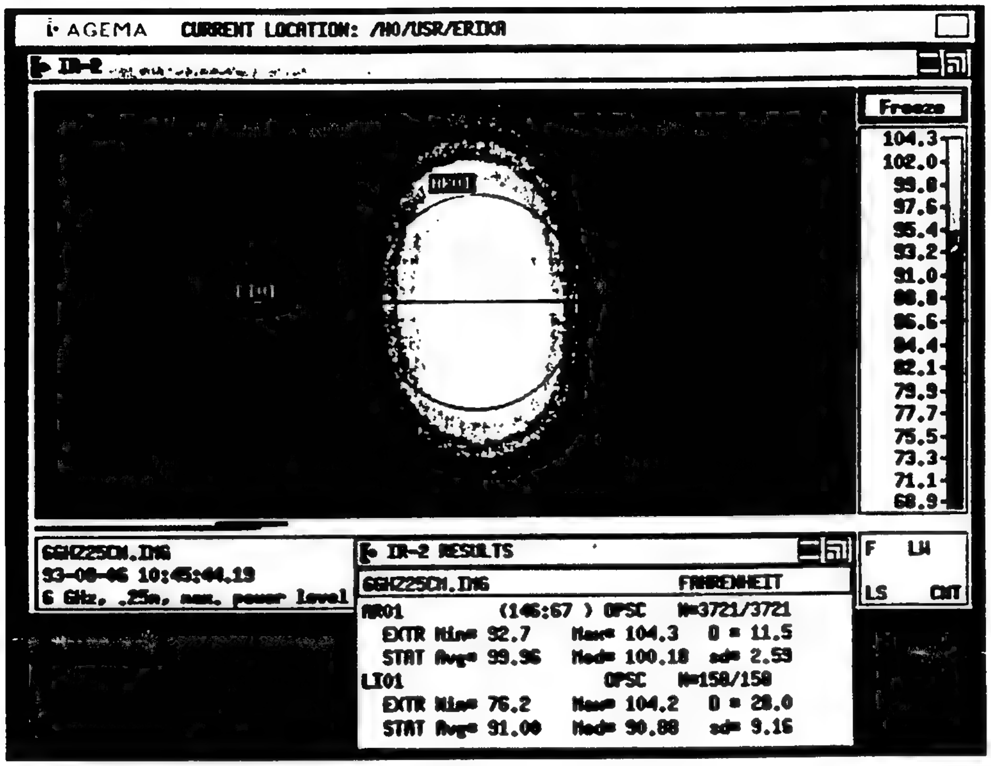
```
6GHZ_THR.IMG
93-08-13 10:04:23.01
6 GHz, .25m, Threshold temp.
```

IR-2 RESULTS

```
6GHZ_THR.IMG                      FAHRENHEIT
AR01          (127:74 ) OPSC    N=3721/3721
  EXTR Min= 65.3    Max= 66.2    D = 0.8
  STAT Avg= 65.78   Med= 65.77   sd= 0.13
LI01                  OPSC    N=158/158
  EXTR Min= 65.3    Max= 66.0    D = 0.8
  STAT Avg= 65.66   Med= 65.71   sd= 0.19
```

Figure 2 IR THERMOGRAPH 100 OHMS/SQ, 25 cm, 6 GHz, NO RF

This process continued until the maximum power level was reached. Figure 3 shows a thermograph of the temperature distribution on a detector with a resistivity of 100 ohms/sq, separation distance of 25 cm, and 6 GHz at the maximum power.



```
6GHZ25CM.IMG
93-08-06 10:45:44.19
6 GHz, .25m, max. power level
```

```
IR-2 RESULTS
6GHZ25CM.IMG                            FAHRENHEIT
AR01            (146:67 ) OPSC    N=3721/3721
  EXTR Min= 92.7     Max= 104.3   D = 11.5
  STAT Avg= 99.96    Med= 100.18  sd= 2.53
LI01                      OPSC    N=158/158
  EXTR Min= 76.2     Max= 104.2   D = 28.0
  STAT Avg= 91.00    Med= 90.88   sd= 9.16
```

Figure 3 IR THERMOGRAPH 100 OHMS/SQ, 25 cm,6 GHz, MAX. POWER

The transmit power is calculated using equation 1,

$$P_{ant}=P_m+L_{coupler}-L_{cable} \qquad \text{EQUATION 1}$$

where:

P_{ant} is the power at the horn antenna (dBm)

P_m is the peak power meter reading (dBm)

$L_{coupler}$ is the loss of the coupler and attenuator (dB)

L_{cable} is the loss of the cable from the coupler to the horn (dB)

To change the transmitted power from (dBm) to milliwatts equation 2 was used.

$$P_t = 10^{P_{ant}/10}$$ EQUATION 2

where:

P_t is the transmitted power in milliwatts

The incident power density is calculated using equation 3,

$$P_d = P_t * G / 4 * \pi * R^2$$ EQUATION 3

where:

P_d is the power density (mW/cm^2)

P_t is the transmitted power from the horn (mW)

G is the antenna gain

R is the separation distance between horn and detector (cm)

The recorded temperature for each power level was subtracted from the ambient to obtain the temperature difference or delta temperature. Shown in figure 4 is a plot of incident power density versus delta temperature, with a resistivity of 100 ohms/sq, and separation distance of 25 cm, at 4, 5, and 6 GHz.

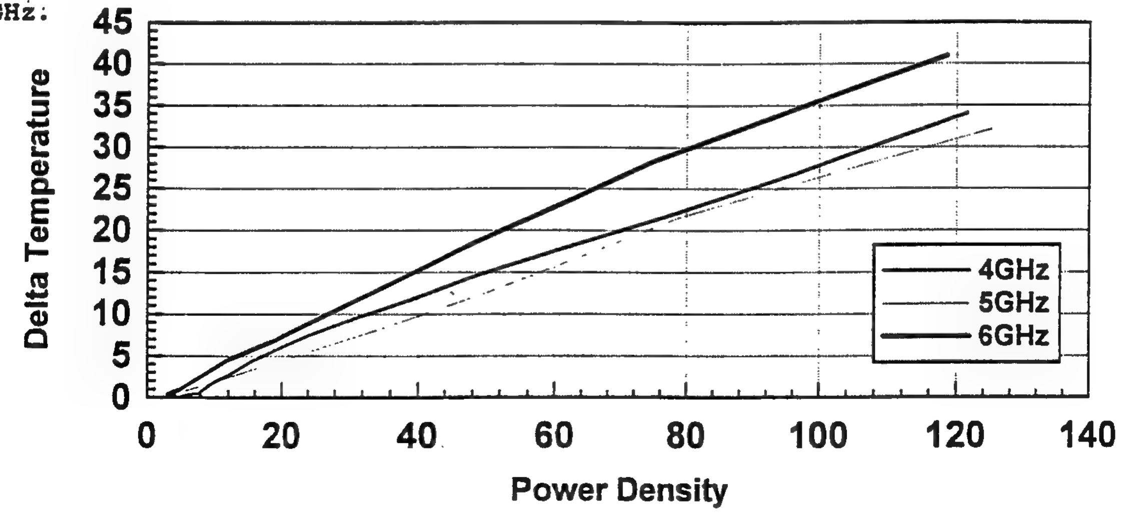


Figure 4 PLOT OF 100 OHMS/SQ, 25 CM, 4, 5, 6 GHz

Conclusions

RF measurements were performed over the frequency range of 2-8 GHz, and for separation distances between the horn antenna and the detector of 25-100 cm. The resistivities of the RF detectors used in the experiment were 100 ohms/sq, 1000 ohms/sq and 1500 ohms/sq. The RF detector with a 100 ohms/sq resistivity achieved a higher temperature increase. This indicates that lower resistivity RF detector absorbed more RF energy and may be more efficient for low RF field measurements.

Incident fields were radiated on the 100 ohms/sq detector at a separation distance of 25 cm for the frequencies 4, 5, 6 GHz. The data is shown in figure 4, at 6 GHz the maximum delta temperature was 41 degrees, at 4 GHz it was 34.1 degrees, and at 5 GHz it was 32.2 degrees. This indicated that the 100 ohms/sq detector was slightly more sensitive at 6 GHz although the slope of the curves were similar.

The results of this work indicate that two-dimensional temperature distribution measurements can be performed fairly quickly for a wide range of incident fields and power levels. The increase in temperature (delta temperature) above the ambient seems to be a linear function of the RF fields. The sources of error in this experiment were caused by amplifier fluctuations in transmitted power, small variations in the separation distances between the horn and detector and the duration of the dwell time of the measurement to obtain a maximum temperature reading.

References

[1] M. F. Seifert, A. J. Pesta, "Infrared Measurements of Electromagnetic Fields", IEEE Dual-Use Conference, May 1993

[2] Agema Infrared Systems, Thermovision 900 Series User's Manual.

Appendix A - Equipment

Equipment	Manufacturer	Model	Serial Number
TWT Amplifier	Logimetrics	A600/c opt 1 and 3	452
Solid State Amp	M/A-COM Microwave	LAB 1-4080-1	7147B-001
Spectrum Analyzer	Hewlett Packard	8562A	2809A02501
Peak Power Meter	Wavetek	8502	1812627
Synthesized Sweeper	Hewlett Packard	8341B	2730A00879
Raham Antenna #2	AEL	5200	104
Thermovision 900	Agema		
Power Detector	Wavetek	17267	46-00205
Coupler	Microwave Corp.	3004-30	01002
Attenuator	Weinschel	23	482

TRIANGULATION USING FM STATIONS

Holly L. Grabowski

Final Report For:
High School Apprenticeship Program
Rome Lab

Sponsored by Air Force Office of Scientific Research

August 1993

Triangulation Using FM Stations

Holly L. Grabowski

Abstract

Global Positioning Systems have been developed for the purpose of determining the coordinates of desired locations on the globe. However, GPS's are costly and an inexpensive replacement has long been sought after. Triangulation is a technique often utilized in finding the coordinates of an unknown location. This technique is currently being implemented to determine the coordinates of an antenna at Hanscom Air Force Base in Bedford, Massachusetts. Local FM stations are the tools which will allow this triangulation to take place. A direction finding device can be used to determine relative azimuth angles of the signals of specific FM stations. These angles, along with the known coordinates of the FM stations, allow for the location of the coordinates of the antenna in question. A computer program was written to process the angles and coordinates, average the results of several triangulations, compensate for errors and noisy signals, and come up with the best coordinates for the antenna. The computation of the coordinates of the antenna at Hanscom AFB alone is not an amazing discovery. However, if these coordinates can accurately be found by the method described above, there is potential for an economical replacement for Global Positioning Systems using existing resources such as FM frequencies.

Triangulation Using FM Stations

Holly L. Grabowski

Introduction

Over the past few centuries triangulation has been used to more accurately determine the size and shape of the earth. Its main use is for determining the geographic positions of different points on the globe. Triangulation is modernly used to locate coordinates of a site if reference coordinates are known. FM radio stations can be used as reference locations. Antenna equipment was used to detect different radio signals and determine the azimuth angle of transmittance (clockwise with respect to north). After this data was obtained, trigonometry was used to find the coordinates of the location in question. However, noise often distorts the readings and a single triangulation could have a large margin of error. That is why many triangulations should be done with different FM stations and the coordinates that have significant error should be discarded. I developed a program in the computer language C which implemented the preceding calculations, to triangulate from multiple antenna measurements at various FM frequencies, using averaging and error-reduction, to determine the unknown coordinates of the antenna.

Methodology

USAF Rome laboratory (at Hanscom Air Force Base) is developing a phased-array antenna which will be utilized in this triangulation experiment. The direction finding antenna can provide information on transmitted plane wave signals. It will have many receivers, each of which will measure the phase and amplitude of the signal. If the signal is received at any angle except 0 or 180 degrees, the phase will not be constant among the receivers. The phase difference results from the time lapse between detection in successive receivers. The angle is found according to the following equation:

$$\Delta\Phi = \frac{2 \cdot \pi \cdot d \cdot sin\theta}{\lambda}$$

Where:

$\Delta\Phi$ = change in phase

d = distance between receivers

θ = angle we are trying to find

λ = wavelength of radio signal

The preceding formula was used by another apprentice (Adam Maloof) to make a program which would calculate the angles. The phases could be entered and the program would calculate all possible angles and then average them to come up with the best angle for each station. These angles were the necessary data for my program, which did the actual triangulations. However, since the lab is still developing the antenna which will acquire the phase data, our two computer programs will not be fully tested until after our apprenticeships have ended. We decided to acquire some data on our own in order to test the triangulation program. Dr. Hugh Southall (Adam Maloof's mentor) had built a simple device which was of use to us in collecting our data. This device consists of a transistor radio and two receivers. The radio can be tuned to the desired frequency and, if the signal is coming from any angle other than 0 or 180 degrees, the phases will be different on each of the receivers and a humming sound will be produced. The apparatus can be manually rotated until the humming stops. When such a null is reached, we know that the phases are the same on both receivers and the signal is therefore perpendicular to the bar between the receivers. With a compass we can then measure the angle of the signal. Although this device is simple, we hoped that it would allow us to get some rough angles for the radio stations from which we could triangulate to calculate the approximate coordinates for the base.

Discussion

To acquire some of the data for this project, I had to first determine which FM frequencies came in clearly at Hanscom Air Force Base. I also had to listen to each station until the call letters for that station were broadcasted. Once the frequencies and call letters were in hand, I needed to identify the coordinates of each station. I attempted to look them up in the library without success; I also tried calling the Federal Communications Commission where I was told that they don't provide that information. Finally I ended up in contact with a company called ITS. They researched the information and mailed me the coordinates of 24 local radio stations.

Once I acquired the coordinates, I had the data that I needed to write my program. My task was to write a C program which would read in the angles and call letters of various stations, and then access the coordinates of each of those stations so that triangulations could be performed with each possible combination of two stations. It would be easy enough to sit down with a piece of paper, locate two stations on a grid, draw a line through each station at the predetermined angle, and find the intersection of the two lines. The process of getting a computer to do this is not as simple and obvious, but a computer program has the potential for higher accuracy, speed, and multiple iterations.

When writing a program it is essential that the programmer make things as simple as possible for the computer. For example, if a person has a set of coordinates and a slope for a line, the equation of the line is easily produced. If the person has two such equations, he/she could solve them simultaneously for the intersection. However, a programmer cannot tell a computer to solve simultaneously. He or she must separate the equations into simple elements for the computer to process sequentially. Let us take the following two equations as an example:

$$y - y_1 = m_1(x - x_1)$$

$$y - y_2 = m_2(x - x_2)$$

The only way the the computer can solve these equations is if the programmer develops and states the following formula in code:

$$x_{final} = \frac{y_2 - m_2 \cdot x_2 + m_1 \cdot x_1 - y_1}{m_1 - m_2} \tag{0.1}$$

$$y_{final} = m_1(x_{final} - x_1) = y_1 \tag{0.2}$$

Where:

x_1 and y_1	= coordinates of a point on the first line
m_1	= slope of first line
x_2 and y_2	= coordinates of a point on the second line
m_2	= slope of second line
x_{final} and y_{final}	= coordinates of intersection of the two lines

The preceding formulas are ones that the computer can interpret. It will automatically substitute in the correct values for x_1, y_1, x_2, y_2, m_1, and m_2. The x and y values are the coordinates of two radio stations, but the slopes (m_1 and m_2) are not immediately accessible. I tested different lines and angles until I realized that the slope of the line is equal to the tangent or cotangent of it's angle (depending on which quadrant it is in). A slope function was added to the program in such a way that it would be recalled each time a new slope needed to be calculated from an angle. Two slopes along with two sets of coordinates are enough data for a single triangulation.

Perhaps the most interesting (and frustrating) aspect of this project entered at this point. What happens when there are *more* than two sets of coordinates? If the data was precise, all of the triangulations would intersect at the exact same point, and this point would be the solution of the equations of the multiple lines. However, problems such as inaccurate devices, noise on the readings, and scattering cause there to be a margin of error. Therefore all possible intersections must be found. The radio stations must be paired in all possible combinations of two. The number of intersection points equals the number of combinations of two. For example, if all of the 24 possible radio stations were used, the following would be the total number of combinations:

$$_{24}C_2 = \frac{24!}{2!(24-2)!} = 276$$

The question is, how do I make the computer compare each radio station with each other one? If I knew how many radio stations were going to be used, I could use the brute force method and write all of the comparisons into the body of the program. However, the program must be general so that any number of radio stations can be used. The easiest program to write would be one that only accepted two stations at a time. This way the user would have to find all of the combinations and average the results. However, I knew that this was impractical and I needed to find a technique that would make the program more user friendly.

The use of counters and loops turned out to be the keys to my problem. I needed a way to identify all of the sets of call letters and angles as different pieces of data which could be accessed individually. The following code served this purpose:

```
struct data {
   char name[4];
   double angle;
}values[25];
printf(''Enter the number of stations that you wish to use:'');
scanf(''%d'',&numb);
for(i=0; i<numb; i++){
   printf(''Enter the call letters of a station'');
   scanf(''%s'', &values[i].name);
```

```
        printf(''Enter the angle of that station'');
        scanf(''%lf'',&values[i].angle);
    }
```

First, I created a structure named *data*. A structure is a technique used to organize variables so that they have meaningful names and are easily understood by the person maintaining the code. In this structure, *values* is declared as an array of 25 elements, *name* is an array of 4 characters, and *angle* is a double (a double is a real number). Possible variables from this structure are $values[n].name$ and $values[n].angle$, where n is a number from 0 to 24. This structure is very useful in my code because if I have two pieces of data identified as $values[3].name$ and $values[3].angle$, it is clear that they are for the same radio station since they are both in the same array location (location defined by the 3 in brackets). After I defined the structure, I had the user enter the number of stations that he/she wished to use (number of stations = *numb*). The number of stations determines the number of iterations of the loop. At the start of each iteration, the integer i is incremented by one (this is represented by $i++$) until i is one less than the number of stations. During each iteration, a new pair of call letters and corresponding angle is asked for and then entered. The sets of call letters and angles of the stations are declared as arrays in the structure. As i is incremented, the loop declares the input data as a new element in its respective array. After the loop has terminated, this data is stored and can be accessed later. If, for example, you wanted to do something with the final angle that had been entered, you would have the program call up $values[numb-1].angle$. The computer would substitute in for *numb* and find the value that had been stored in $numb-1$ location of the array.

Once the call letters and angles were initialized, I needed a way to call up the coordinates that corresponded to the specified stations. I created a function for each station which contained the coordinates for that station. These functions perform no calculations. They simply contain data that the program needs to read. However, the program does not necessarily need the coordinates for *every* station. It only needs the coordinates of those stations which the user specifies. Therefore, once I found a way for the program to access only the necessary coordinates, I had all of the tools necessary to write the most important component of my program - the part that found all the combinations of two stations and calculated the triangulation for each combination. Again, loops and counters were used in the following abbreviated code:

```
1 for (a=0; a<(numb-1); a++){
2     for (b=(a+1); b<numb; b++){
```

```
3          if (strcmp(values[a].name, ''wmbr'') == 0){
4             wmbr(x1, y1);
5          }
6          else if(strcmp(values[a].name, ''wers'') == 0)
           .

           .
7          if (strcmp(values[b].name, ''wmbr'') == 0){
8             wmbr(x2, y2);
9          }
10          else if (strcmp(values[b].name, ''wers'') == 0)
           .

           .
11          xfinal = (see equation 0.1);
12          yfinal = (see equation 0.2);
      }
   }
```

This section of code contains a loop within a loop. In the outside loop a starts off at 0 while b is incremented from 1 to $numb - 1$. Then a is incremented to 1 while b goes from 2 to $numb - 1$, and so on until $a = numb - 2$ and the only option for b is $numb - 1$. Therefore inside the loop the code can be written for the computer to compare the station in the a array location to the station in the b array location. The result is that every station is paired with every other station. Each iteration does a new combination. The call letters of the two stations referred to by the nested loop are compared to all possible call letters. If there is a match, the function for that station is called and the coordinates are accessed. In lines 3, 6, 7, and 10 the command *strcmp* appears. It stands for 'string compare'. If the string that the user entered for *values*[*a*].*name* was "*wmbr*", then *strcmp*(*values*[*a*].*name*, "*wmbr*") would equal 0 (this means that the two strings are identical). In this case the function *wmbr* would be called and the coordinates of that station would be $(x1, y1)$. If the user entered "*wers*" for *values*[*b*].*name*, the function *wers* would be called and the coordinates of that station would be $(x2, y2)$. The slopes for the two stations called up in each iteration will be calculated as m_1 and m_2 by the slope function. This means that the program is continuously redefining the global variables x_1, y_1, x_2, y_2, m_1, and m_2. The redefinition of the variables at each iteration allows for there to be one general equation for x_{final} and y_{final} (see equations 0.1 and 0.2). The functions that have been called up substitute specific local variables for all of the global variables in the general equations 0.1 and 0.2. At each iteration of the nested loop new stations will be called up and new values for x_{final} and y_{final} will be found until all possible intersections have been calculated.

After all possible intersection points have been calculated, one question still remains. What method should be used to combine this data? All of the points could be arithmetically averaged, but a single point with a large amount of error could significantly throw off the final average. As was stated earlier, the data for this program is going to be acquired after my apprenticeship ends. I had no data to give me any indication of the expected degree of error. Therefore, I had to find a way for my program to be flexible and allow the user to decide how much error he/she wanted the computer to remove. In order to implement this feature, I first had the computer average all of the triangulations and print out the averaged coordinates. Then, I coded for a prompt to appear and ask for what percent the user wanted eliminated. For example, if the user entered 30, the program would drop all intersection points more that 30 percent different from the average. The computer would then average the remaining values and print out that result. The program can be run multiple times with different percentage deviation cutoffs until the user finds a percentage that seems to fit the data well without getting rid of too many or too few of the intersections.

At the completion of the main sections of my program, I began to fine tune it so that it was easier to use. For example, I coded the program in such a way that it would read a file containing all of the data. This way the user could change one piece of data in the file and rerun the program without having to reenter every single piece of data. However, the program had yet to be tested. As I mentioned in the methodology, another apprentice and I hoped to get some rough data through the use of a device built by Dr. Hugh Southall. We brought the apparatus out to a field, tuned it into a known station, and tried to find the null in the humming sound. Unfortunately, we had to try to hear the humming sound over the radio station that we were tuned in to. In addition, the null often existed over a span of ten to twenty degrees. The data that we recorded had too much error to even give us a rough idea of our coordinates and we were forced to abandon this attempt to test my triangulation program.

Conclusion

I have shown that it is possible to write a program which can determine the coordinates of a specific antenna using its phase measurements at many FM frequencies to triangulate with error reduction techniques. The implication of this successful project is that economical replacements for Global Positioning Systems can be developed which will utilize existing resources such as FM frequencies.

References

1. ITS - a company under contract by the Federal Communications Commission. Phone Number 202-857-3836

2. Technical discussions with Terry O'Donnell, USAF Rome Laboratory, Electromagnetics and Reliability Directorate, Hanscom AFB.

3. Technical discussions with Major (Dr.) Jeffery Simmers, USAF Rome Laboratory, Electromagnetics and Reliability Directorate, Hanscom AFB.

4. Technical discussions with Dr. Hugh Southall, USAF Rome Laboratory, Electromagnetics and Reliability Directorate, Hanscom AFB.

5. Whitmore, George D. Advanced Surveying and Mapping. Scranton: International Textbook Company, 1949. p. 1-4.

A STUDY OF ARRAY ANTENNAS AND NEURAL NETWORKS FOR FUTURE APPLICATION IN DIRECTION FINDING

Adam Maloof

High School Apprentice

Lexington High School

251 Waltham Street

Lexington MA 02173

Final Report for

AFOSR Summer Research Program

Rome Laboratory

AFOSR Sponsorship

August 1993

A STUDY OF ARRAY ANTENNAS AND NEURAL NETWORKS FOR FUTURE APPLICATIONS IN DIRECTION FINDING

Adam Maloot
High School Apprentice
Lexington High School

Abstract

Various types of array antennas were the subject of this study. The investigation focused on the patterns in wave data observed when the value of theta (the angle of arrival i.e. the angle at which a transmitter's phase front strikes an element of an array antenna) was increased or decreased. Phase and amplitude patterns were observed and plotted for selected values of theta with an eight element phased array lab antenna. Computer programs were written to simulate and plot 1) theta for different element spacings of a two element array, and 2) the phase difference between elements of a three element array for different thetas. Ultimately, these programs will serve as the foundation for developing an advanced direction finding mechanism. The data collected from the lab array will be used to train a neural network capable of describing theta for any given wave data. This neural network might also find application as an efficient direction finding device.

A STUDY OF ARRAY ANTENNAS AND NEURAL NETWORKS FOR FUTURE APPLICATION IN DIRECTION FINDING

Adam Maloof

Introduction

In recent years neural networks have emerged as powerful and efficient pattern recognition devices Although network training is often a laborious task requiring large data sets, direction finding techniques, where the detection of hidden data patterns is crucial, may be an ideal application for neural networks Direction finding has long been a woefully inaccurate science especially when using affordable techniques However with a receiving antenna array FM reciever the geographical coordinates of two FM radio stations and a neural network to interpret the FM wave data precisely one could triangulate and find one s longitude and latitude with astounding accuracy Such a direction finding technique finds promising application in vehicle navigation

Methodology and Apparatus

The process of gathering the tools to run a successful direction finding operation requires both theoretical information generated by computer plots and experimental information obtained through laboratory observation The methodology and apparatus used in this study are discussed in two sections below

I After an introduction to the literature regarding antennas, direction finding, and neural networks computer programming became the focus of this investigation Following trials with Matlab™ and Mathematica™, the investigator chose Matlab™ as his engineering support because of it's superior programming and debugging capabilities To learn Matlab™ and Emacs, a Unix system text editor, the investigator plotted a number of RMS Error graphs for his mentor Once in control of Matlab™ the task of analyzing wave data for changes in theta (the angle where a transmitter's phase front strikes an element of an array antenna began

First a plot of theta for various phase differences and element spacings of a two element array was generated As can be seen in Figure 1 an inverse sine wave was the result It was noted that as element spacing increased so did the horizontal shrinkage of the sine wave These data proved helpful but somewhat inconclusive for most arrays have more than two elements

Secondly a plot of phase difference for changing theta of a three element uniformly spaced array was generated Given a three element array a reference element and the position of a radio station one can deduce certain patterns in ideal wave data as theta increases or decreases as is described in Figure 2 (i e as the position of the radio station or the antenna changes) A plot consisting of two sine waves each depicting a different elemental phase difference was the result (figure 3) Though many arrays are not uniformly spaced this three element array yielded useful insights into ideal wave data patterns caused by changes in theta In a realistic application a similar program would allow the investigator to measure theta and phase difference at all the elements relative to the reference element making the process more efficient Once accurate data from each element are obtained the process could be repeated for a second radio station A triangle of points would then be formed (triangulation) between the investigator and the two radio stations One s location could then be determined geometrically with great precision

II While computer programing the investigator also conducted a series of laboratory observations with lab technician Airman First Class Jim Rouse The laboratory apparatuses consisted of an eight element phased array antenna The array was situated in a measurement bay partially lined with absorber so that specular reflection could be held to a minimum A fixed source attenuated at 110 dB generated almost no radiation while a horn mounted on a moving armature transmitted approximately four one-hundreths of a watt (note a CB transmitts 3-4 watts) to the array The moving source moved along a track in front of the array in one degree intervals from negative sixty degrees to positive sixty degrees and the attenuation was a variable ranging from 0 dB to 70 dB Phase and amplitude were measured for each element of the array for each degree that the moving source travelled A switching matrix allowed individual data collection for each element The procedure was repeated at 0 10 20 30 40 50 60 and 70 dB of attenuation

The data from the lab experiment outlined above were collected daily and fed into a local area network of computers for analysis Phase and amplitude plots were generated during each experimental trial so that excess noise and scatter could be detected It was found that as the source generated less power (i e attenuation of 50, 60, or 70 dB's) the data began to reflect more noise contamination

Conclusion

Ultimately the phase and amplitude readings from each trial that had been transferred to the network will be used to train a neural network In the laboratory phase measurements collected across the array deduce the source direction The neural network will be trained with this phase data and the corresponding source direction angles If the neural network is successful, it will be able to determine the direction of a transmitter given the phase data of the wave The phase data of the wave are easily determined with sensitive measuring devices and a variation of the second computer program outlined above

Figure 1: Values of theta for different phi and d

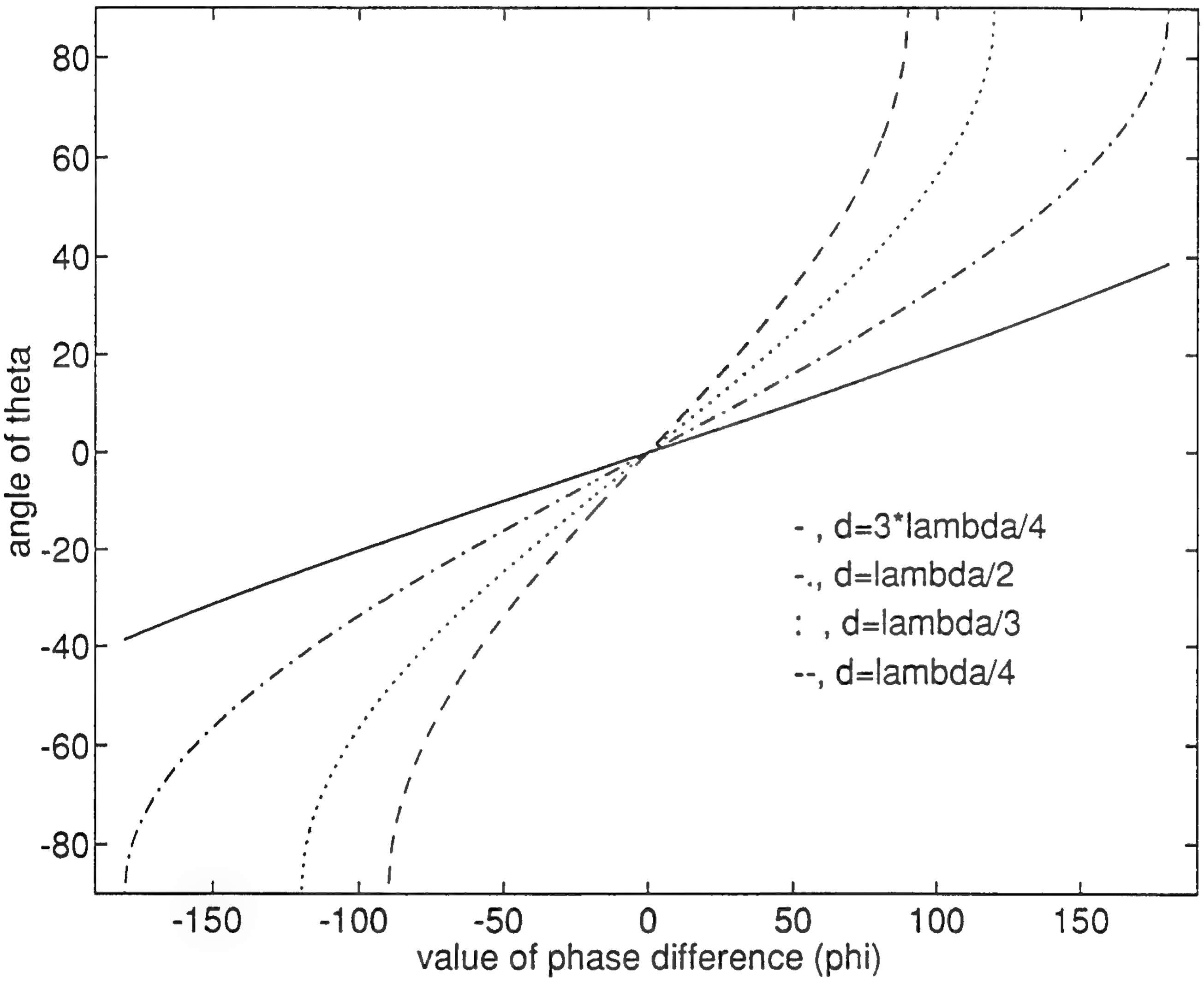


FIGURE 2: THREE ELEMENT ARRAY GEOMETRY

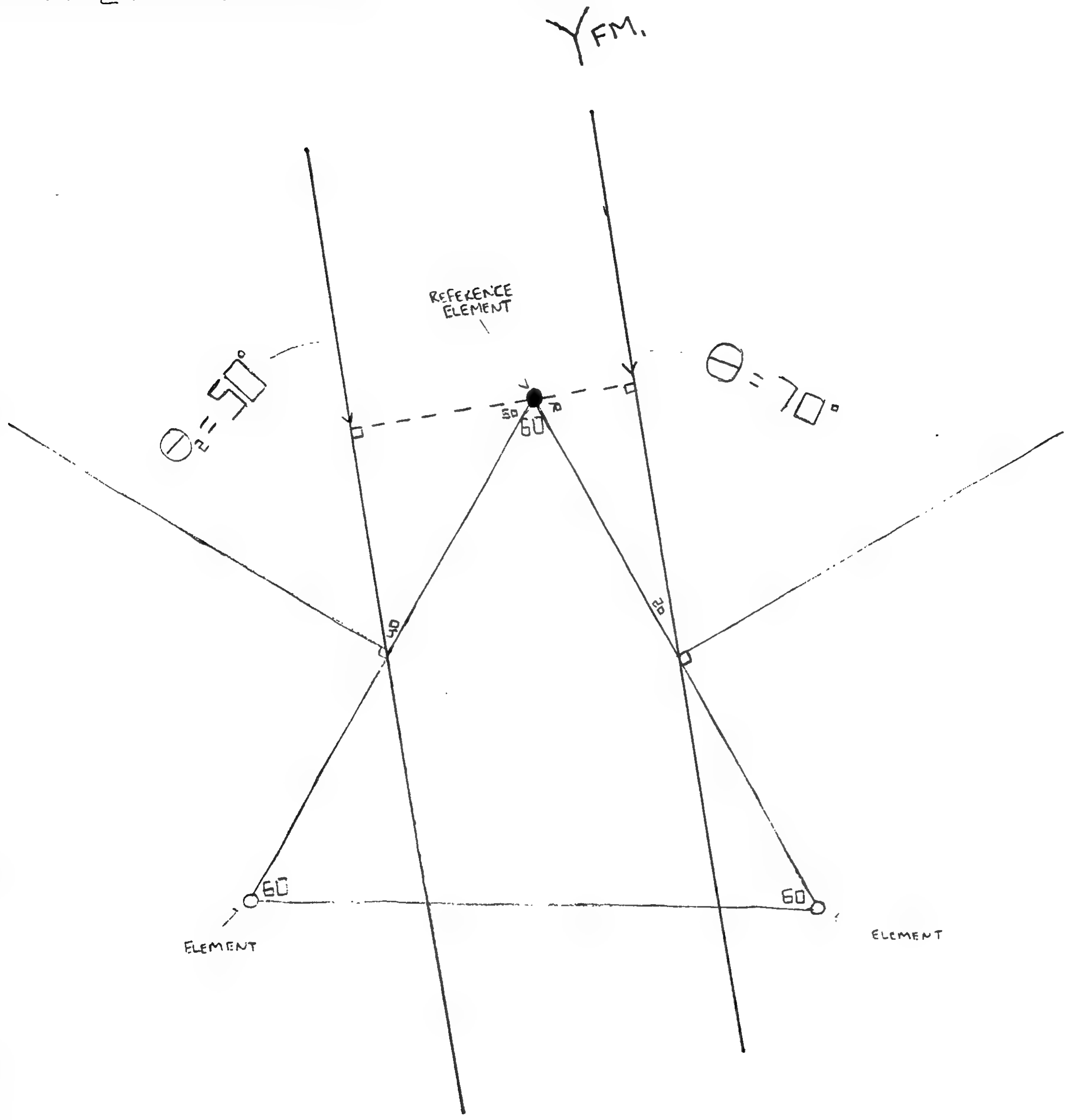
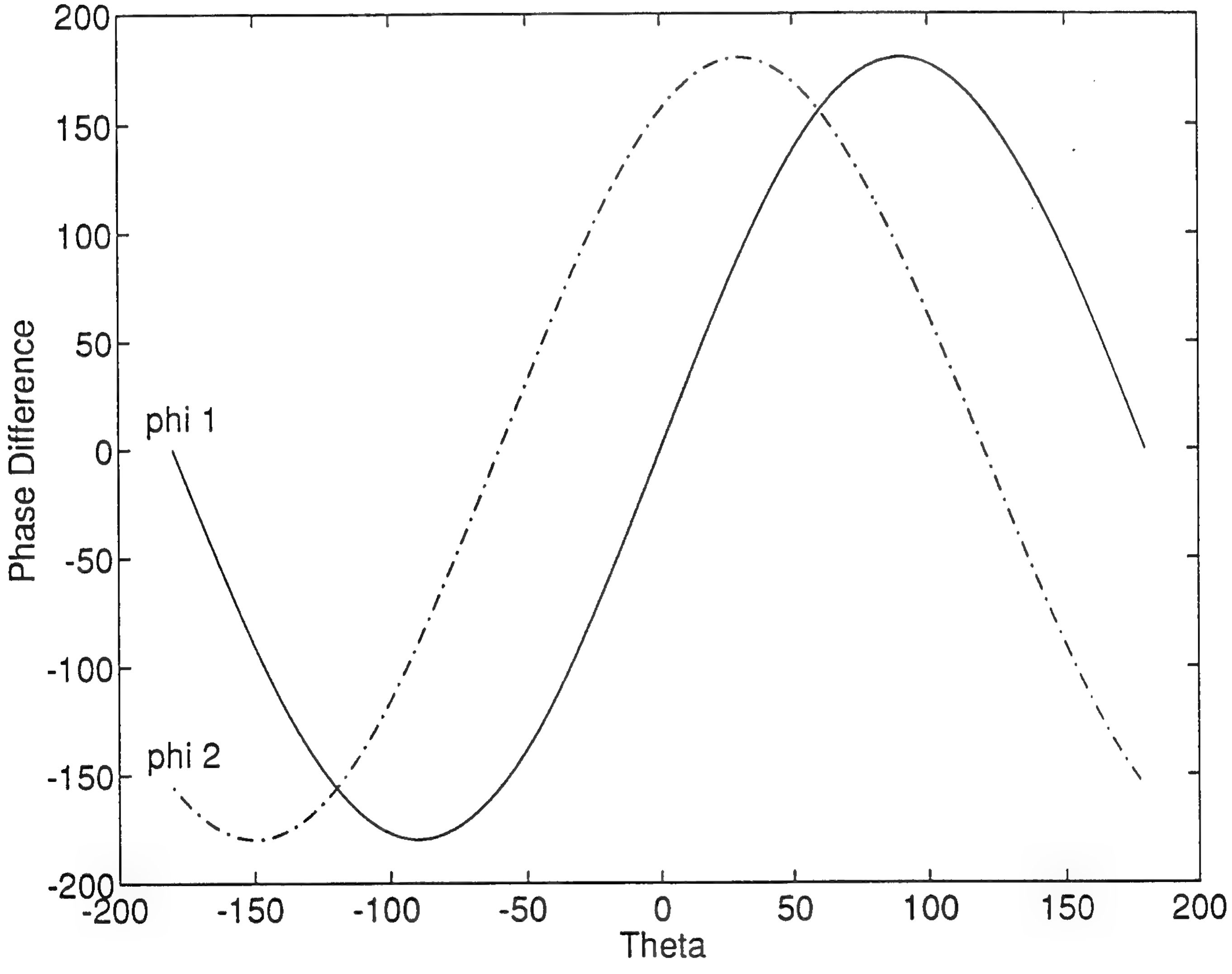


Figure 3: Phase Differences vs Angle of Arrival (theta)



References

Adams Steve Mathematica Demo 11 June 1992

Balanis Constantine A Antenna Theory Analysis & Design Harper and Row Publishers New York 1982

Gray Theodore W and Jerry Glynn The Beginner s Guide to Mathematica version 2 Addison-Wesley Publishing Company New York 1992

Hammerstrom Dan Neural Networks at Work IEEE Spectrum Vol 30 No 6 June 1993 26-32

Hammerstrom. Dan "Working with Neural Networks" IEEE Spectrum volume 30. number 7 July 1993 46-53

Johnson Richard C Antenna Engineering Handbook Third Edition McGraw-Hill Inc New York 1993

Lamport Leslie Latex User s Guide and Reference Manual Addison-Wesley Publishing Company, Reading, 1986

Li Hsueh-Jyh et al Nonuniformly Spaced Array Imaging In *IEEE Transactions on Antennas and Propagation* Vol 41. No 3. March 1993

Matlab User's Guide Mathworks. Inc Natick 1992

O Brien. M and H Southall Memorandum Monopulse Ratio with Electrical Errors 16 March 1993

O Donnel. Terry ERA Computer & Network Overview 6 April 1993

O'Donnel. T et al Neural Beamforming for Phased Array Antennas In *Proceedings of the 1992 Antenna Applications Symposium (Robert Allerton Park)* Griffiss AFB, NY. September 1992 USAF HQ Rome Laboratory

Phased-Array Radar System Design Course Notes Georgia Institute of Technology. Atlanta. GA. 13-16 April 1993

Simmers J A and T O'Donnel Adaptive PBF Neural Beamforming In *Proceedings of the IEEE Mohawk Valley Section Command Control Communications and Intelligence (C3I) Technology & Applications Conference* pages 94-95 IEEE. June. 1992.

Stallman, Richard M. Gnu Emacs Manual, seventh edition, version 18.58 for Unix Users. September 1992.

Tsoi, A.I. Multilayer Perceptron Trained Using Radial Basis Functions. *Electronics Letters* 25(15): 1296-1297, 14 September 1989.

The Effects of Wet and Dry Silicon Dioxide Passivation Etchants on Aluminum Metallizations for Integrated Circuits

Nathan Terry
Senior
Clinton Senior High School

Final Report for:
AFOSR Summer Research Program
Rome Laboratory

Sponsored by:
AFOSR High School Apprenticeship Program

Aug 1993

The Effects of Wet and Dry Silicon Dioxide Passivation Etchants on Aluminum Metallizations for Integrated Circuits

Nathan B. Terry
Senior
Clinton Sr. High School

Abstract

The capability of six surface analysis techniques to determine the effects of passivati[illegible] etchants on aluminum metallizations was examined. A buffered hydrofluoric acid wet etchant and a plasma tetrafluoromethane dry etchant commonly used to etch silicon dioxide passivation layers were studied. Atomic Force Microscopy was shown to have the highest topographical resolution while Auger Electron Spectroscopy and ESCA both gave significant elemental surface information.

Introduction

When integrated circuits were first developed. electromigration was discovered to be a serious problem. Techniques and processes such as adding a passivation layer were then developed to minimize electromigration effects, but, as the size of integrated circuits continued to decrease to their present levels, electromigration is again a serious problem. The voltage necessary to operate integrated circuits has remained constant, while the cross sectional dimensions of the interconnects through which the current flows has decreased. This change in scaling has caused an increase in current density that has resulted in electromigration damage. Because the metallizations are so small, sometimes less than a micron wide, electromigration creates voids large enough to cause significant changes in the interconnect resistance, and, in some cases, grow large enough to cause a physical opening. Another possible failure mode is the development of protrusions that may become large enough to cause a short circuit. It is of great importance to find a way to retard electromigration effects. One of the first steps in doing this is to characterize electromigrated interconnects. However, the passivation layer, which is often composed of SiO_2, interferes with most materials characterization techniques. Therefore, it is necessary to find a method that will remove the passivation layer without damaging the aluminum metallization beneath it. While, to a first approximation, plasma etching has been considered a nondestructive technique for removing silicon dioxide passivation layers, new analytical techniques such as Atomic Force Microscopy (AFM), which have increased surface sensitivity, have resulted in the re-examination of this assumption at the atomic level.

Methodology

Because of the microscopic size of integrated circuit interconnects, it was impractical to use them for this experiment. The ideal dimensions for a sample in Electron Spectrometry for Chemical Analysis (ESCA) is 10 mm by 3 mm. Aluminum metallization samples of these dimensions were fabricated at the Rochester Institute of Technology in a four step process. First. a 350 Å layer of SiO_2 was thermally grown on a 1.5 cm diameter silicon wafer. and then 700 Å of aluminum were sputter deposited onto the SiO_2. Photolithography was then used to pattern the aluminum into strips 5.2 mm in width. The samples were then cut into 1.85 cm squares.

a. Plasma Etching

Plasma etching is generally considered a nondestructive technique for removing silicon dioxide passivation layers from aluminum metallizations. An aluminum metallization sample was randomly chosen to be plasma etched. This sample was placed into a Tegal Plasma Etcher and etched using CF_4 at 5 psig and 50 Watts for one to five minute intervals for a total of 30 minutes. Between each period of etching, of which there were seven. the sample was taken out of the Plasma Etcher and profiled using a Tencor Instruments Alpha-Step 200 profilometer with the profile stylus set to eleven milligrams of force. This instrument operates by physically dragging a stylus over the sample. The pressure on the stylus was fixed, so it followed the contours of the sample. The vertical motion of the stylus was recorded as a function of the horizontal distance traversed using the starting point as the origin. Each profile was compared to the one taken before the etching began. The sample was then etched again for another period of time. Once the etching was completed, the sample was photographed at 500x using an optical microscope. Because the resolution of the optical microscope was insufficient to allow an effective grain size determination, the plasma etched sample and the control sample were then examined with Scanning Electron Microscopy (SEM) and Atomic Force Microscopy (AFM).

Other characterizations were performed on the sample by ESCA and Auger Electron Spectrometry (AES). Both ESCA and AES are surface analysis techniques. Both of these processes depend on a high energy source. either an electron beam in the AES or X-rays in ESCA, to excite an atom. In ESCA, a photoelectron is released. In AES a core electron is ejected and a second electron from a higher energy shell of the same atom drops to that lower energy level to fill the vacancy. The energy from that transition is transferred to a third electron, and finally the Auger electron is ejected from the atom. In both cases the energy of the resulting electron is characteristic of the excited atom or element, which makes it possible to determine the source atom or element. The AES was used for depth profiling by sputtering off part of the specimen with an argon ion beam. After a number of atomic layers were sputtered off, the surface was characterized. and then the sputtering was resumed. This process continued until the silicon surface was reached.

b. Wet Etching

A plasma etchant, a wet etchant was also used. Two specimens were etched by placing them into a basket constructed from a four ounce, 125 ml polypropylene bottle. A pair of scissors and a razor were used to cut the sides off the basket and a screwdriver functioned as an awl to produce the holes in the side and bottom (see fig. 1). A sample was placed in this basket and immersed in the etchant. Once the etching was complete, the basket was removed and the etchant drained. The basket and the sample were then immersed in de-ionized water to remove any remaining etchant.

The advantage to this method was that it was very safe. There was little danger of dropping the sample into the etchant. However, a disadvantage was that the sample was not always immersed in a uniform manner. One of the samples actually floated on the surface of the etchant for approximately one second. Another disadvantage was the existence of a two second margin of error in measuring and controlling the duration of the etch. This was due to the time involved in draining the basket and the uncertainty of determining exactly when the sample became totally immersed. However, if the sample had been etched using tweezers, drops of the etchant clinging to the sample would have had the same effect.

These samples were etched in a common wet oxide etch, one part NH_4F and seven parts HF by volume. One was etched for five seconds, the other for two seconds. Neither sample could be ESCA profiled because there was so little surface area left. Similarly, AES depth profiles could not be done because the remaining sample was not thick enough due to the etch. However, visual and optical micrsocsopy observations as well as surface analysis by AES were done on the sample.

<u>Results</u>

a. Plasma Etching

The AES depth profile of the control sample confirmed that the aluminum metallization was 700Å thick (see fig. 2-1). The depth profile of the plasma etched sample showed that the stripe remained approximately 700 Å thick (see fig. 2-2). On the other hand, virtually all of the 350 Å layer of SiO_2 was etched by the plasma etchant (see fig. 2-3, 2-4). These results showed that although the SiO_2 was etched, with the depth resolution of AES, the aluminum metallization was not etched.

AES surface analysis revealed the presence of oxygen, aluminum and carbon were present on the surface of both the control and the plasma etched samples, while the plasma etched sample also contained fluorine in large amounts (see fig. 3-1, 3-2). A more detailed surface analysis using ESCA revealed that there was roughly the same percentage of carbon in the two samples, but the amount of oxygen in the plasma etched sample decreased by about twenty percent. Although the aluminum concentration of each sample was roughly 10%, the elements bonded to the aluminum changed greatly. The aluminum compounds in the first few atomic layers of the control sample were 93.4% Al_2O_3 and 6.6% pure aluminum. In the plasma etched sample, the aluminum compounds in the top atomic layers were 49.2% AlF_3, 41.3% Al_2O_3 and 9.5% pure aluminum (see fig. 3-3, 3-4).

The topographical changes in the sample were examined by optical microscopy, AFM and profilometry. Optical microscopy revealed a thin black layer at the border between the aluminum and the silicon dioxide. Further examination by SEM confirmed that this layer was an insulator, which supports the conclusion that this was a layer of Al_2O_3. This layer not only formed on the sides of the aluminum, but also on other places on the surface. This layer was visible on the control sample, though it was several orders of magnitude thinner. The SEM also revealed small holes in the etched sample that were much more numerous than they were in the control sample.

AFM showed pronounced changes in the surface topography of the samples. The most noticeable change was in the grain boundaries, which were much more pronounced in the sample that had been etched (compare fig. 4-1 and fig. 4-2). This may mean that the grain boundaries were preferentially etched. The profilometer revealed no change in the etched sample because it was not sensitive enough to detect the topographical changes.

b. Wet etching

Both visual and optical microscopic examinations revealed that the two samples etched in the wet oxide etchant had virtually all of the aluminum removed. Signs of a rapid chemical reaction, such as small bubbles coming from the sample, were distinctly visible to the naked eye during the etching process.

Conclusions

The buffered wet etchant was extremely ineffective in leaving the aluminum stripe undamaged. It was a very fast etch, and would be far more useful for etching macroscopic samples. The CF_4 plasma etchant did etch the aluminum surface, and greatly enhanced the surface grain boundaries.

One of the least useful instruments in this experiment was the profilometer. It did not have the resolution required to give changes in the plasma etched sample and gave only minimal changes in the wet etched sample (see fig. 5-1, 5-2).

Optical microscopy was only marginally effective at identifying changes in the plasma etched sample. It did have enough magnification to show that something formed on the boundary between the SiO_2 and the aluminum. However, the SEM was also able to detect this layer. It also was able to detect the localized holes in the surface of the plasma etched sample but did not have the resolution necessary to observe widespread changes in the surface.

AES and ESCA proved to be very useful in determining the chemical changes on the surfaces of the samples. However, AES depth profiling did not have enough resolution (the depth is only accurate to one significant figure) to determine how much of the aluminum metallizations had been etched by the dry etchant. A greater degree of precision would make AES depth profiles more informative.

AFM proved to be very useful in analyzing changes in the surface of the aluminum. Unlike the profilometer, it had the resolution required to give detailed quantitative data on the changes the aluminum surface underwent due to etching. This instrument could also prove very useful in future studies of the removal of passivation layers using other etchants, such as more dilute HF etchants or different plasma etchants.

Appendix

Etching Basket

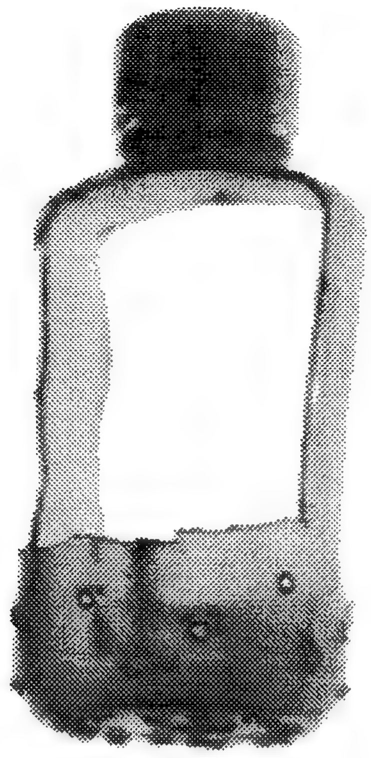


fig. 1

AES Depth Profile of Control Sample

AES Depth Profile V/f Alternating 23 Jul 93 Species: Al2 Region: 6 Area: 1 Sputter Time: 20.00 min

File: alum21 top surface As received sample #10

Scale Factor: 82.370 kc/s Offset: 0.000 kc/s Ep: 10.00 kV Ip: 0.0983 uA

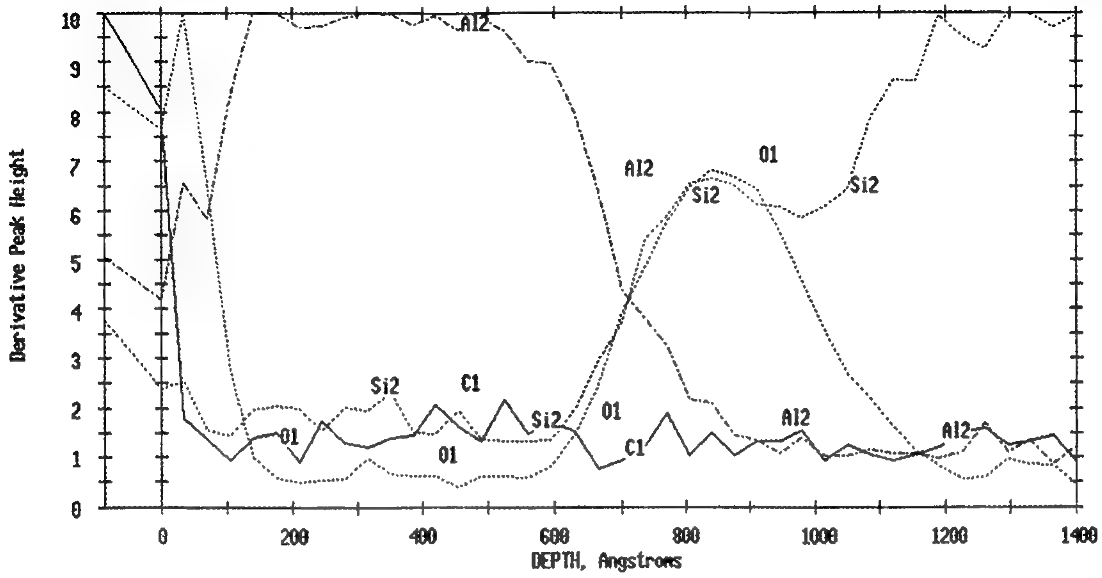


fig. 2-1

AES Depth Profile of Plasma Etched Sample

AES Depth Profile V/f Alternating 22 Jul 93 Species: Al2 Region: 6 Area: 1 Sputter Time: 20.50 min

File: alum7 Top surface plasma etched 30 min sample #1

Scale Factor: 110.000 kc/s Offset: 0.000 kc/s Ep: 10.00 kV Ip: 0.1133 uA

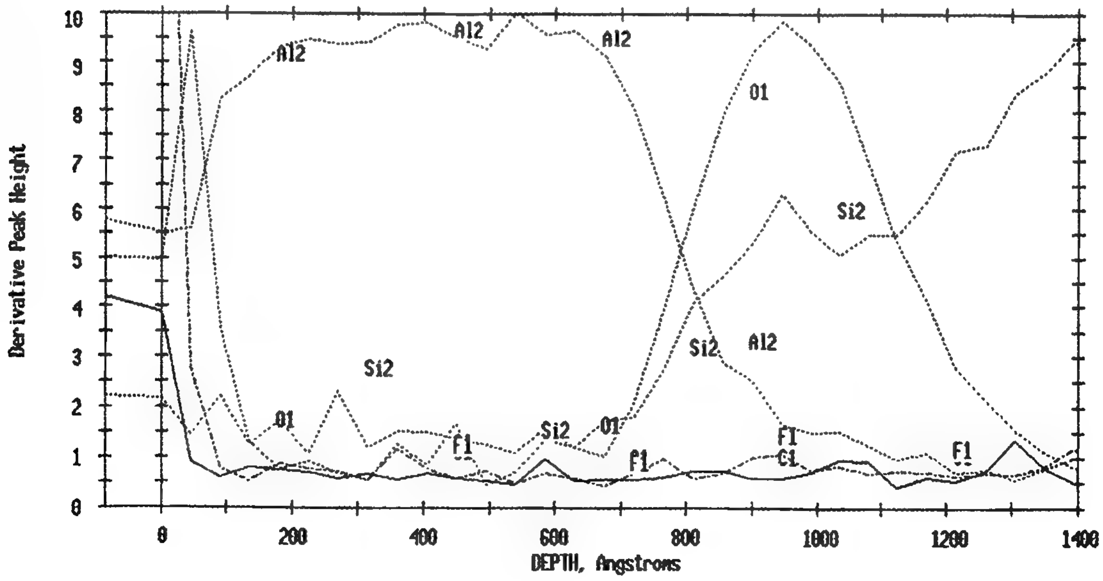


fig. 2-2

AES Depth Profile of Plasma Etched Sample

AES Depth Profile V/f 23 Jul 93 Species: Si3 Region: 4 Area: 1 Sputter Time: 7.00 min

File: alum23 top surface As received sample #10

Scale Factor: 87.873 kc/s Offset: 0.000 kc/s Ep: 10.00 kV Ip: 0.0969 uA

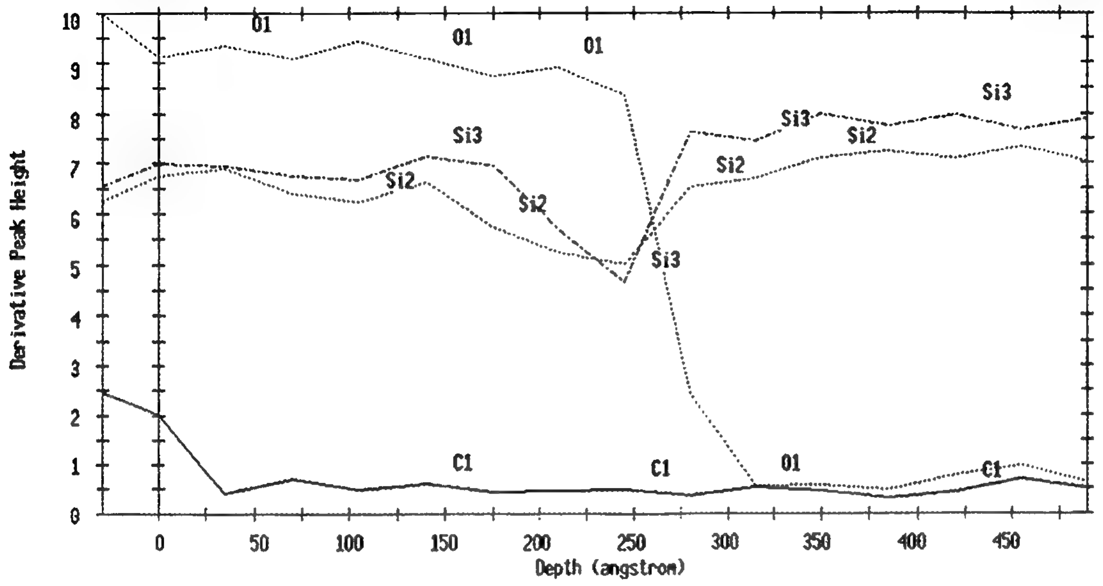


fig. 2-3

The O1 line shows the location of the SiO_2

AES Profile of the Plasma Etched Sample

AES Depth Profile V/f 22 Jul 93 Species: Si3 Region: 4 Area: 1 Sputter Time: 7.00 min

File: alum5 Top surface plasma etched 30 min sample #1

Scale Factor: 61.981 kc/s Offset: 0.000 kc/s Ep: 10.00 kV Ip: 0.1125 uA

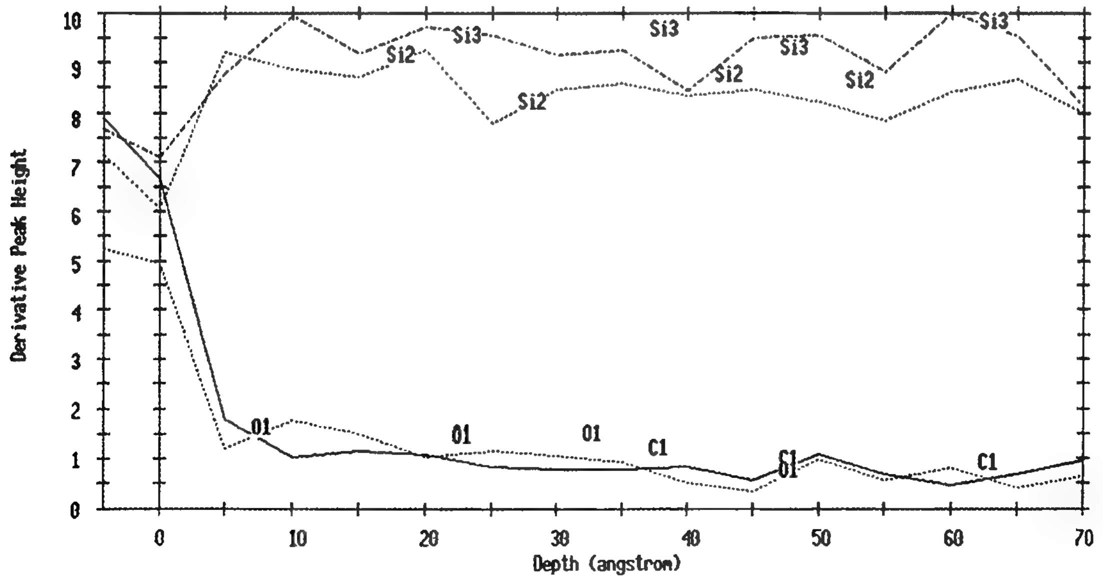


fig. 2-4

AES Chemical Analysis of Control Sample

AES Survey V/f 23 Jul 93 Area: 1 Acquisition Time: 2.50 min

File: alum20 top surface As received sample #10

Scale Factor: 250.872 kc/s Offset: -1315.400 kc/s Ep: 10.00 kV Ip: 0.0967 uA

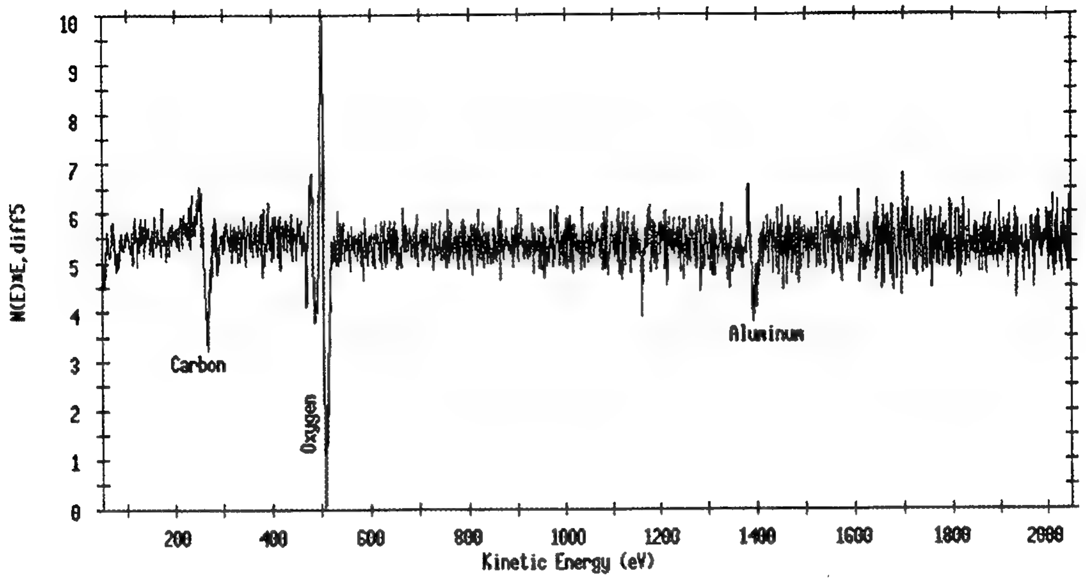


fig. 3-1

AES Chemical Analysis of Plasma Etched Sample

AES Survey V/f 22 Jul 93 Area: 1 Acquisition Time: 2.50 min

File: alum6 Top surface plasma etched 30 min sample #1

Scale Factor: 423.393 kc/s Offset: -2272.283 kc/s Ep: 10.00 kV Ip: 0.0253 uA

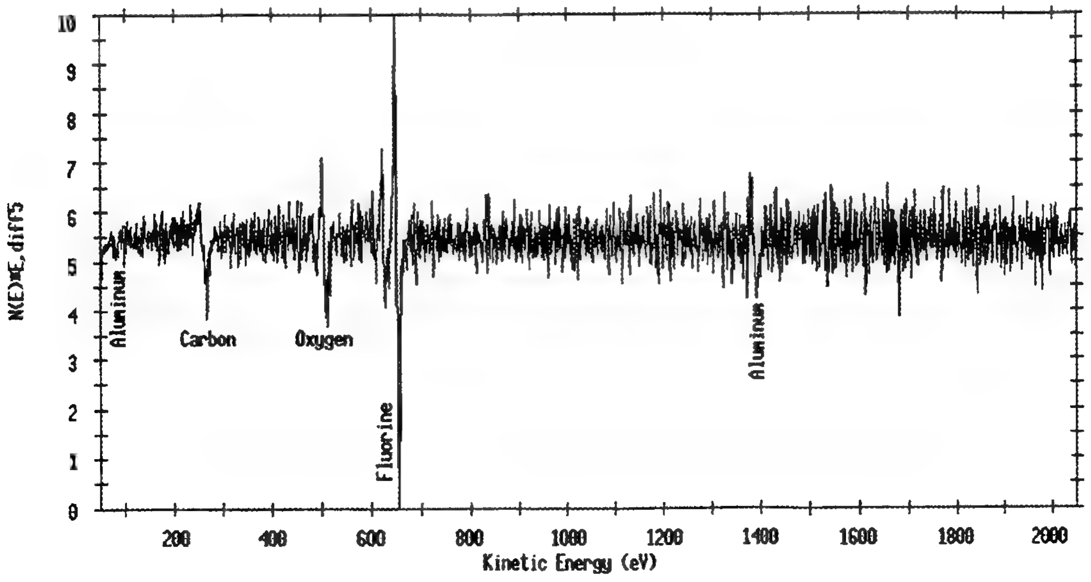


fig. 3-2

ESCA Chemical Analysis of the Aluminum Content of Control Sample

ESCA CURVE FIT 7/23/93 ANGLE= 90 deg ACQ TIME=3.34 min
FILE: alu29al1 sample 10, Al, No Etch, clips removed
SCALE FACTOR= 0.024 k c/s, OFFSET= 0.000 k c/s PASS ENERGY= 17.900 eV Mg 100 W

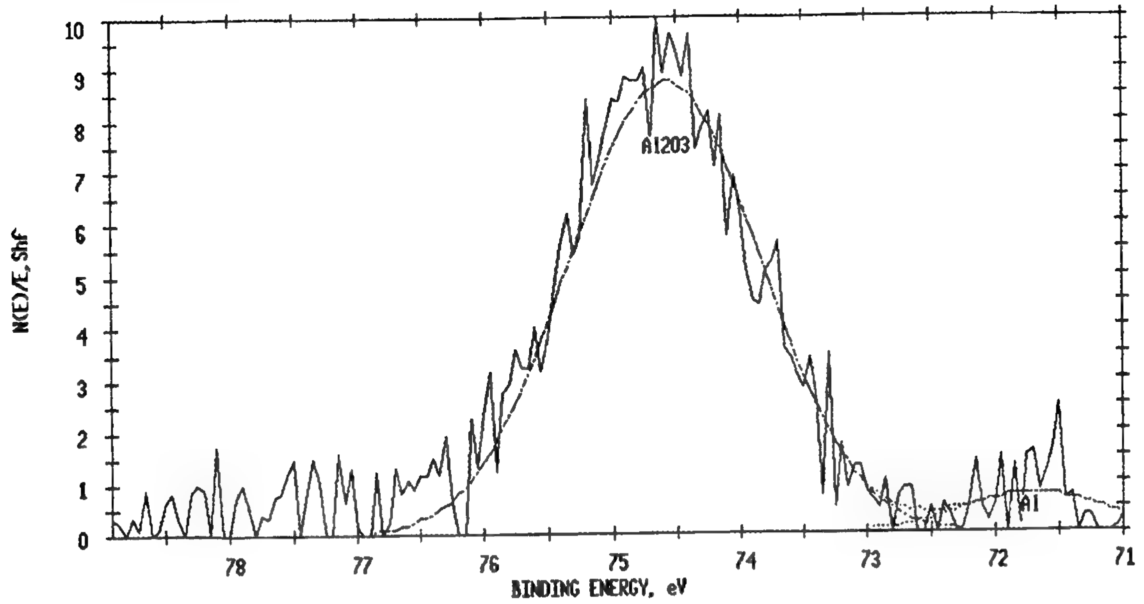


fig. 3-3

ESCA Chemical Analysis of the Plasma Etched Sample

ESCA CURVE FIT 7/20/93 ANGLE= 90 deg ACQ TIME=3.34 min
FILE: alu8al1 Plasma etch 30 min. Alu Sample # 1
SCALE FACTOR= 0.030 k c/s, OFFSET= 0.000 k c/s PASS ENERGY= 17.900 eV Mg 100 W

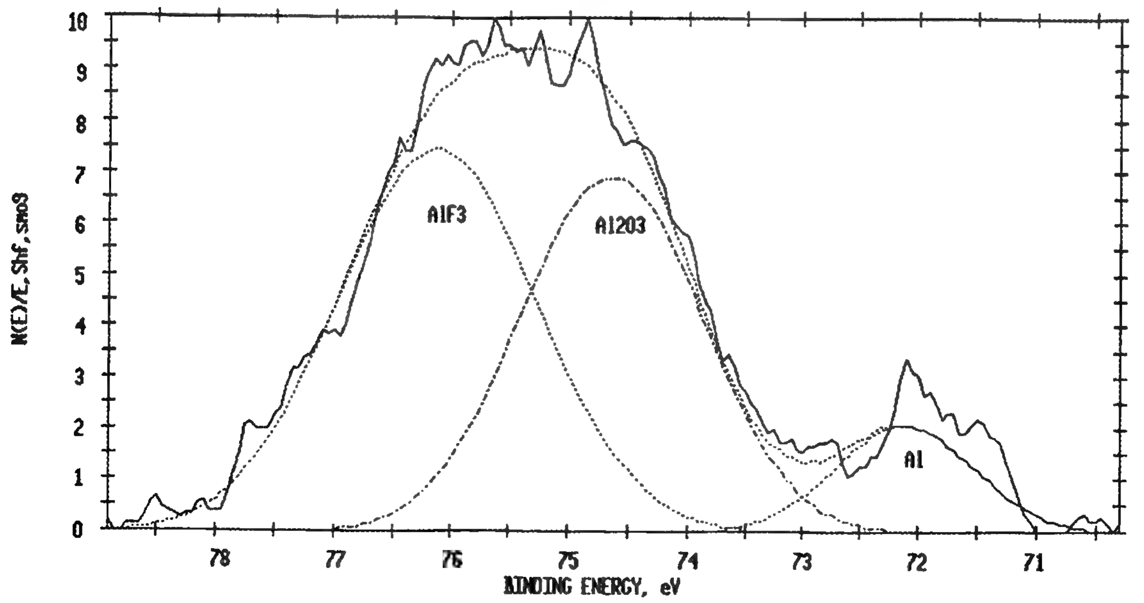


fig. 3-4

AFM survey of Control Sample

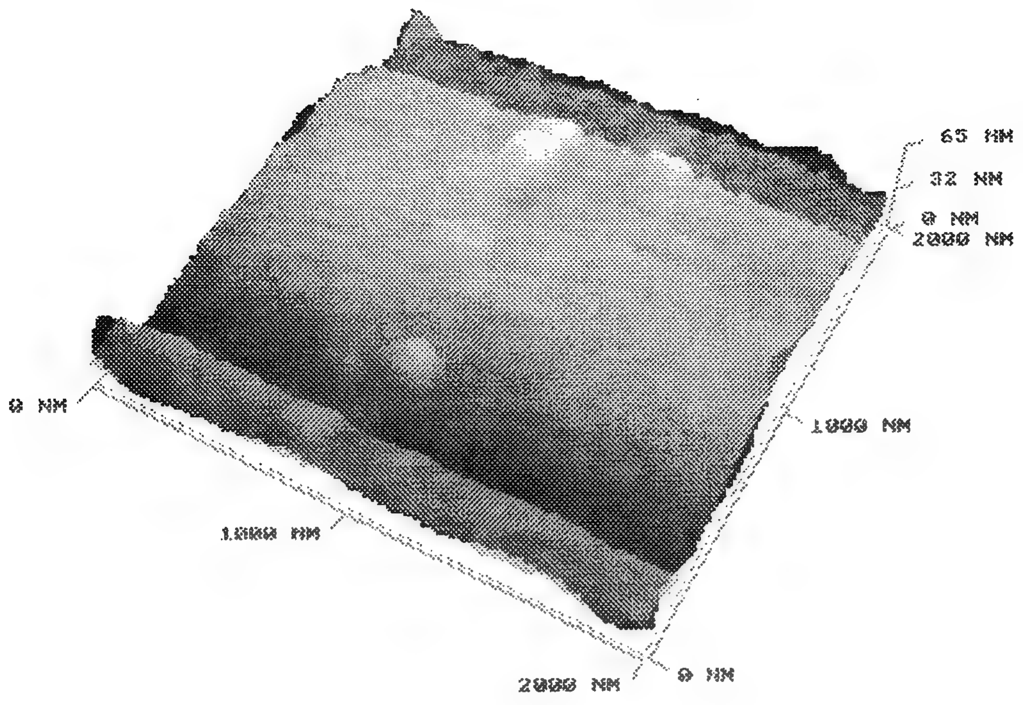


fig. 4-1

AFM Survey of Plasma Etched Sample

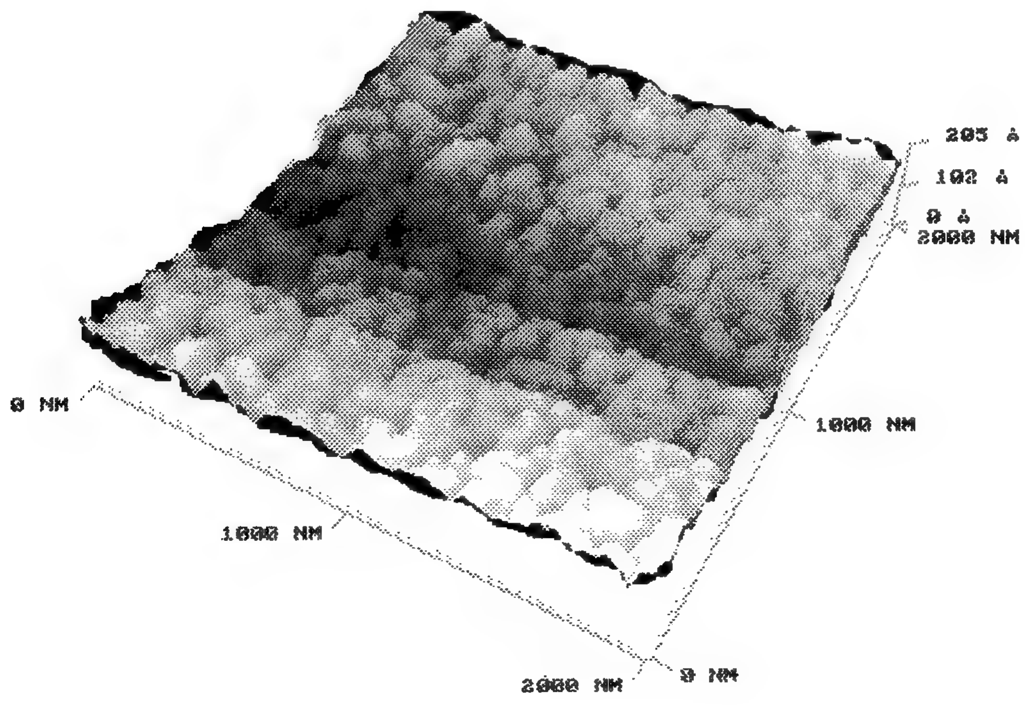


fig. 4-2

Profilometer Survey of Control Sample

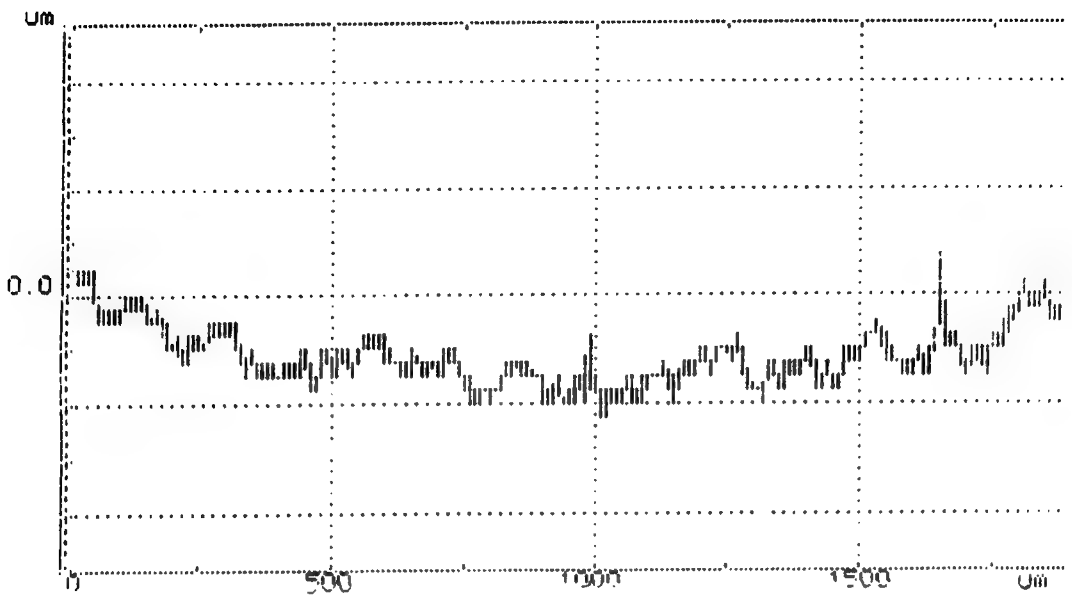


fig. 5-1

Profilometer Survey of Plasma Etched Sample

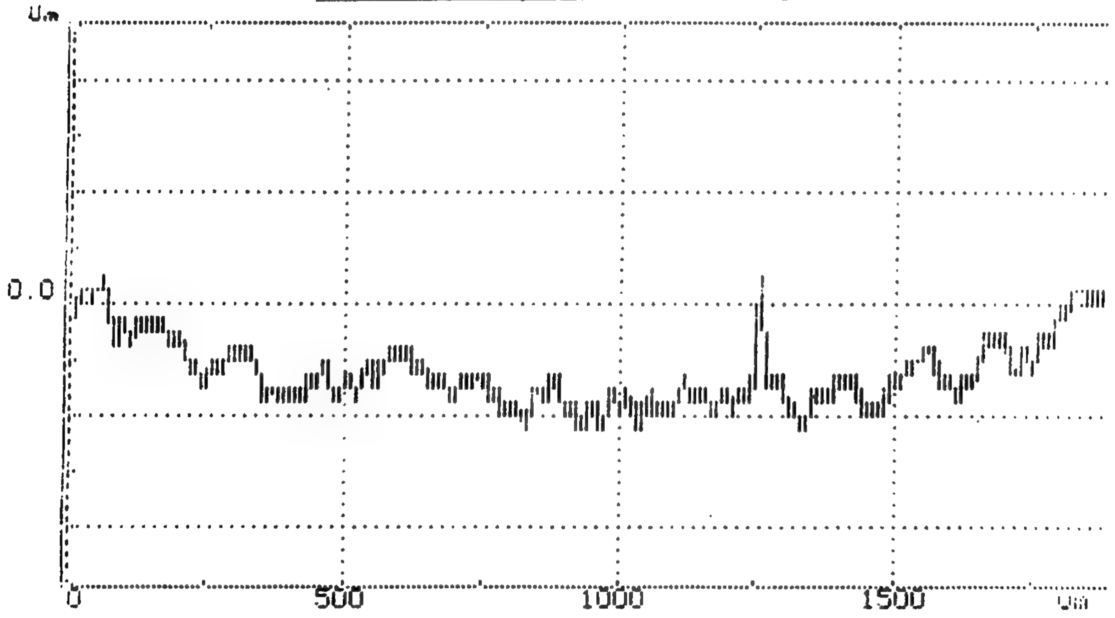


fig. 5-2

How To Establish and Configure a Local Area Network

Stephen A. Antonson

Final Report For

Summer Research Program

Rome Laboratory

Sponsored by:

Rome Laboratory

Griffiss Air Force Base

Rome, New York

August 1993

How To Establish and Configure a Local Area Network

Stephen A. Antonson

Abstract

Interoffice communication needed to be established to increase productivity. The easiest and most effective way to accomplish this was with a LAN (Local Area Network). A variety of network adapters were used to create a thin Ethernet network among five computers. This network was established and maintained by using Microsoft® Windows for Workgroups™.

Introduction

This project had three very distinct parts. First, the hardware needed to be installed (both the network cards and the thin Ethernet cabling laid down). Second, the software needed to be installed and debugged to make sure that both the software and the hardware are compatible. Finally, the computers needed to be integrated together so that they could function as a cohesive unit.

Methodology

First, the network cards were placed in each of the computers. There were three different types of network adapter cards used. There were one Intel EtherExpress™ board, two Western Digital WD/8003EB boards, and two Excelan EXOS 205T boards. The next step in establishing the network was the laying of the cable. Figure 1 outlines the layout of the cable. The layout was designed to allow the least amount of cable usage.

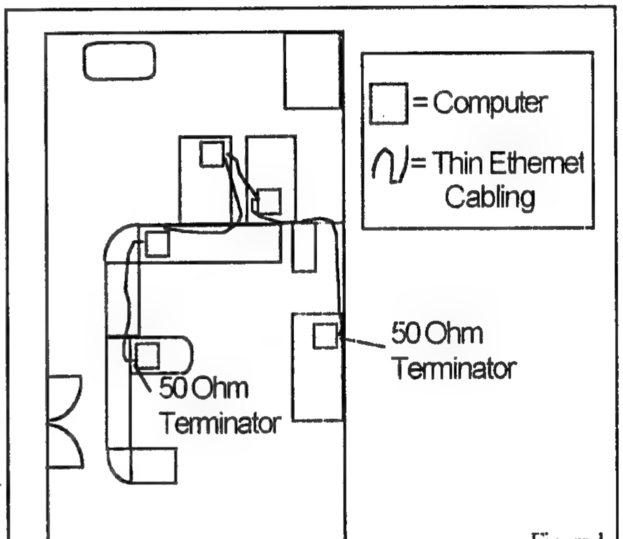


Figure 1

The second step of establishing the network was the software installation and set up. The installation of the software proceeded with little difficulty. The configuring of the software did pose some problems. The Excelan EXOS 205T boards were old and Microsoft® did not provide any specific drivers for the boards as they did for the other two types of boards. To remedy the situation I used a generic driver to provide the interface that Windows for Workgroups™ needed.

The third and final step of establishing the network was the integration of the computers. Integration involved setting up and maintaining a Post Office. This was established so that members of the workgroup could send and receive mail. Establishing the Post Office did pose some problems. When the Post Office was first installed it was installed incorrectly. Subsequently, the Post Office did not function as it was designed. The only way to fix the aforementioned problem was to delete the MSMAIL.INI and the MSMAIL.MMF. These are the two files that mail looks for to initiate the mail program.

Apparatus

The apparatus used were five network boards (various types). Also, four segments of thin Ethernet cabling were used (varying lengths). Fifty ohm terminators were used to complete the link among all the computers. The application software used was Windows for Workgroups™ (five copies). The computers themselves were 386 SX 25Mhz's with four or eight megabytes of RAM.

Results

After the hardware and software problems were resolved, the network functioned without a problem. The network has been up and running for over a week and not a major problem has arisen. Mail and files can be sent and received easily and efficiently. The network improved the efficiency of the office.

Conclusion

The result of this project was a success. Interoffice communication has become much less complicated and it is easier to communicate with co-workers. While working on this project I learned the functionality of Windows® and how it uses its files to function.

I also had another minor project to perform while I was here. I needed to configure two personal computers for a project Calico. The project is classified and this is all the information revealed to me. This project required two identical personal computers. One computer had to stay on the ground while the other computer was placed in the plane. It was my job to ensure that the computers were exactly the same. I made sure that the computers were running the same version of their operating system (DOS version 5.0). I also installed one $3\frac{1}{2}$ inch drive in each computer. I also needed to install one GPIB board in each computer and install the software to allow the computers to interface with the LeCroy Digitizers.

Figure 2 shows the distribution of my efforts during my tour this year. I spent nearly two hundred hours (62.5%) on the networking project. I spent seventy hours (21.9%) working on project Calico. I spent fifty hours (15.6%) on writing this report.

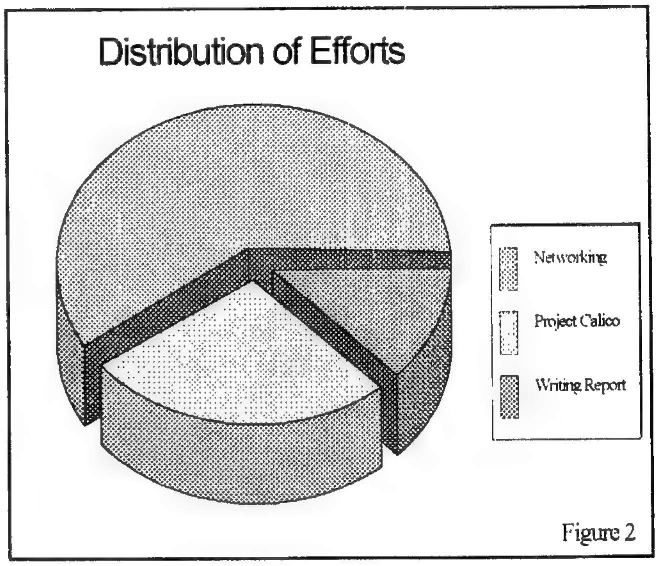


Figure 2

Melissa Hanna's report not available at time of publication.

COMPUTER SIMULATION OF RADIO RECEIVERS

William J. Schatz

High School Summer Apprentice

Final Report for:

AFOSR Summer Research Program
Rome Laboratory

Sponsored by:

Air Force Office of Scientific Research
Bolling Air Force Base, Washington, D.C.

August 1993

COMPUTER SIMULATION OF RADIO RECEIVERS

William J. Schatz
High School Summer Apprentice

Abstract

An AM/FM radio receiver and the receiver's components were simulated through the use of a Macintosh IIx computer in a MATLAB environment. The simulation was performed by writing equations and algorithms defining the various components' functions; real and simulated data was then applied to these equations. A data collection unit was established comprising of a UHF antenna, an Applied Communications Receiver, a preliminary mixer, and an analog-to-digital converter. The collection unit was connected to a CompuAdd 325 computer which saved the digitized data as a binary file. An interface application was written to convert the binary data into a MATLAB compatible form. This data was then "played" using the simulated receiver.

COMPUTER SIMULATION OF RADIO RECEIVERS

William J. Schatz

Introduction

The movement in radio receiver technology has been towards digital receivers. Many receiver designers are replacing analog components with digital components. As the speed of digital signal processing (DSP) rapidly improves, digital components are becoming even more attractive. Analog to digital (A/D) conversions are being made increasingly closer to initial antenna reception so that more digital components can be used in signal processing; faster A/D converters make this possible. Once the radio antenna reception is digitized, the converted reception can be dealt with like any other digital signal.

Digital signal processing has not always been so well-received by the scientific community. Many scientists felt that DSP was just an approximation and simulation of analog signal processing. However, since its inception, DSP has been linked with digital computers. Scientists and engineers saw the advantage of developing and testing signal processing algorithms through digital computer rather than through costly analog hardware. Then, in the 1960's, a whole set of properties and mathematics were developed for DSP after the introduction of the fast Fourier transform (FFT). Algorithms and mathematical procedures were developed that had no translation into analog signal processing. Since the 1960's, digital signal processing has rapidly expanded (Oppenheim and Schafer 4-6).

Digital components hold many advantages over analog components. Digital components, consisting of simple AND, OR, and other logic gates, are less expensive than analog components, consisting of resistors, capacitors, and various non-linear elements such as rectifiers and diodes. Further, analog components are more subject to fatigue than digital components. The biggest advantage that digital systems hold over analog systems is digital's flexibility. Any digital signal can be introduced, given the proper interface, into a digital computer to be processed. Thus, it is quite simple to introduce new modules (mathematical operations, equations, algorithms, tools, etc.) into a signal processing routine. The user has the flexibility to alter the digital signal in any possible manner.

Analog systems do hold advantages over digital systems. Some numerical precision is lost during the A/D conversion process because digital signals are quantized. However, with digital signals, great levels of noise reduction can be achieved because of its discrete time properties reason. The major reason that analog systems are still prevalent is that analog systems are considerably faster than digital systems.

Digital systems have a time delay due to the actual computer processing involved. This delay may be of a small magnitude, but many operators demand real-time capability.

Digital signal processing is becoming increasingly faster. Processing routines can be built directly into hardware. With the increasing interest in computer parallelism and the possible implementation of Residue Number System(to be discussed later), near real-time functioning is in the future for DSP and digital receivers.

The work performed in this project is a demonstration of digital signal processing. Algorithms and equations were established to process simulated and actual digital radio signals. Since the simulated receiver exists in software, it will not be subjected to the fatigue that actual receiver hardware experiences, making it more reliable. In addition, the simulated receiver contains entirely simulated digital components; it should have the capability to greatly reduce noise and data incongruities. Theoretically, because of this digital characteristic, the simulated receiver should have greater clarity than the analog Applied Communications receiver involved in data collection.

Discussion of Goals and Objectives

The initial task was to simulate an AM and an FM receiver in a MATLAB environment on a Macintosh IIx. This was performed by writing equations and algorithms which defined the functions of the various receiver components. MATLAB functions and scripts were created to perform this task. Simulated data was used to test the simulated receiver.

With the simulated receiver established, the next goal was to make a data collection using the data collection apparatus (to be described later). The apparatus originates with a UHF antenna mounted to the building roof. The apparatus can obtain raw data from the antenna (there are a two down-conversions so that the data may be digitized). The data is digitized and stored as a binary file on the CompuAdd 325 computer. However, this PC binary data can not be read directly into the MATLAB environment on the Macintosh.

The next task thus became to read the binary file into the Macintosh. To do this, Lt. Jim Wintermyre created a Macintosh application that allows the PC data to be read into MATLAB. The process is quite fast, taking less than ten seconds for a 200 kilosample conversion. The converted data can easily be introduced into the MATLAB workspace.

The AM/FM simulated receiver is identical to real receivers in form and function. It was hoped that the simulated receiver would be able to demodulate the modulating signal from the data and actually "play" this signal. It was hoped that the receiver would work with relative speed and accuracy.

The final goal was to create new functions and algorithms that allow the user enhanced capabilities to examine the actual receiver data and analyze it. This would include modified functions to "play" a large number of samples. This final task was essential in understanding the characteristics of the raw data and processed data. The data samples were too large for the Macintosh to examine them directly.

Apparatus and Methodology

The project's equipment was composed of two major pieces: the data collection apparatus and the Macintosh IIx/MATLAB application program. The data collection apparatus is shown in Figure 1 on page 14-6. The antenna is a UHF antenna mounted on the building roof. A cable runs down from the roof connecting the antenna to an Applied Communications receiver.

In this apparatus, the receiver is used only as a mixer. The receiver has a frequency range from 20 to 500 MHz. However, all the intermediate frequency (IF) signals leaving the receiver have a carrier frequency of 21.4 MHz, despite their varying radio frequency (RF) signal forms. There are two mixers in the receiver that perform this conversion. Remember that mixers can be used either in up or down conversions.

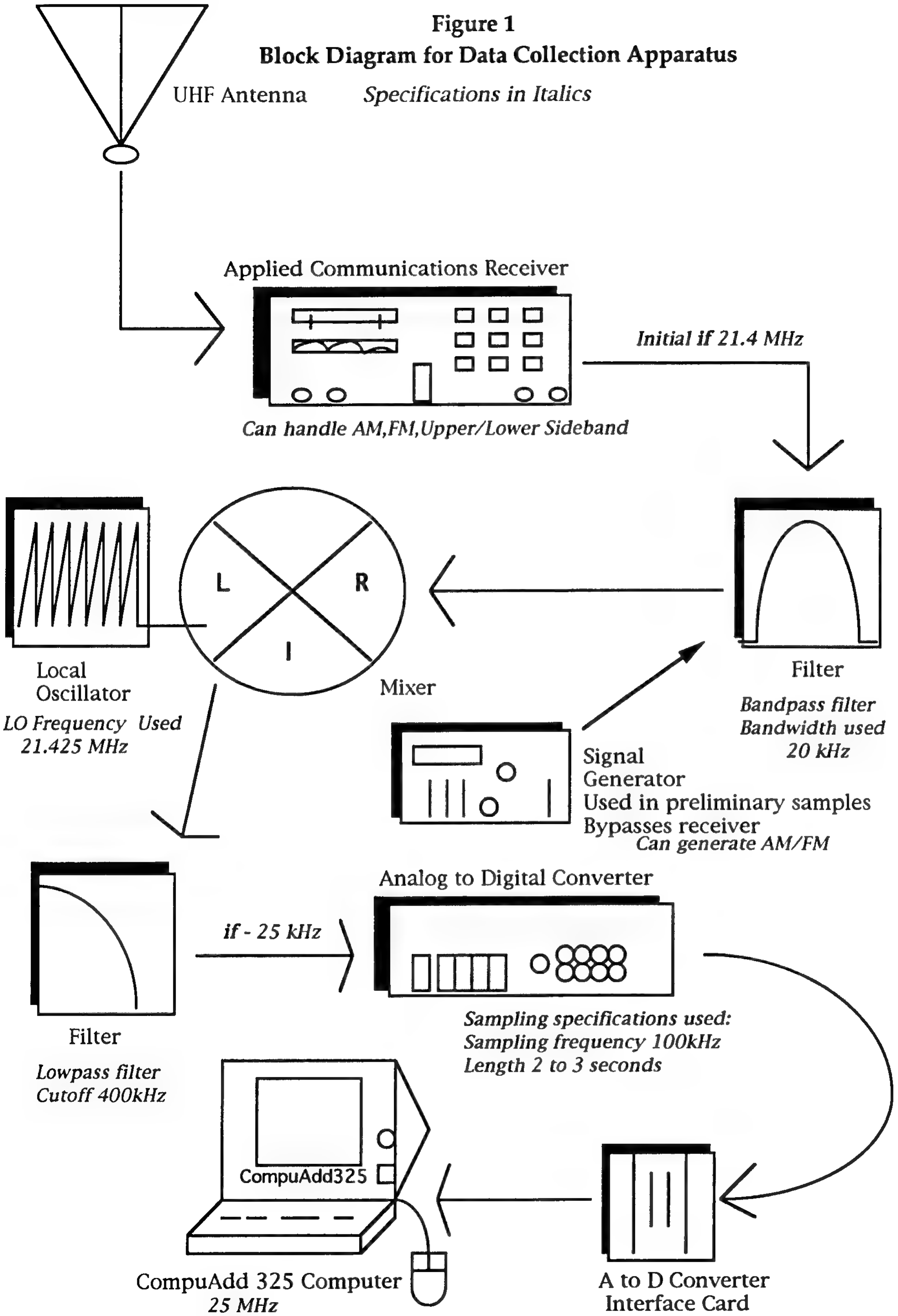
The receiver **IF** then passes through a bandpass filter, limiting the bandwidth. This is to prevent folding (aliasing) later during analog to digital conversion. This filtered **IF** then goes through another down conversion, necessary for A/D conversion. The **IF** has to be down converted because the maximum sampling rate of the A/D converter is 100 kHz. A local oscillator is attached to this mixer and generates a signal with a frequency of 21.425 MHz. After this mixed **IF** is sent through a lowpass filter with a cutoff frequency of 400 kHz, an **IF** with a carrier frequency of 25 kHz remains.

Recall that the difference between the frequencies of the filtered **IF** and the local oscillator was 25 kilohertz. The results follow from the equation:

$$\cos(a)\cos(b) = \frac{1}{2}\cos(a+b) + \frac{1}{2}\cos(a-b)$$

The variables *a* and *b* represent the frequencies of the two signals. The two resultant terms each represent a signal. The first cosine term has a frequency equal to the difference of *a* and *b*. The second cosine term, with a frequency equal to the sum, is removed by the high pass filter. Again, this high frequency signal is removed to prevent folding.

Figure 1
Block Diagram for Data Collection Apparatus



UHF Antenna
Specifications in Italics
Applied Communications Receiver
Initial if 21.4 MHz
Can handle AM,FM,Upper/Lower Sideband
L
R
I
Local Oscillator
LO Frequency Used 21.425 MHz
Mixer
Filter
Bandpass filter
Bandwidth used 20 kHz
Signal Generator
Used in preliminary samples
Bypasses receiver
Can generate AM/FM
Analog to Digital Converter
if - 25 kHz
Filter
Lowpass filter
Cutoff 400kHz
Sampling specifications used:
Sampling frequency 100kHz
Length 2 to 3 seconds
CompuAdd325
CompuAdd 325 Computer
25 MHz
A to D Converter Interface Card

The signal is now ready for analog to digital (A/D) conversion. The sampling rate must be more than twice the highest frequency of the IF to prevent aliasing. Given an intermediate frequency carrier of 25 kHz and a sideband width of 10 kHz (one half the bandwidth), an adequate sampling frequency would be 100 kHz. To obtain a substantial number of samples, the digitizer is allowed to take samples over a period of a second or more. A sample period of two seconds would yield 200 kilopoints.

The specifications for the sample are established on the CompuAdd 325 computer. An "A/D" converter interface card has been installed into the computer. The digitized data is stored in the computer, and a worksheet application allows for initial data analysis and observation. The data can be stored on a floppy disk as a binary file.

However, the data can not be read directly into a MATLAB environment. Lt. Wintermyre stated two reasons for this condition. First, the CompuAdd 325 uses an INTEL microprocessor, while the Macintosh uses a Motorola microprocessor. The computers store binary words in different orders (i.e. the positions of the high and low order words are different) in their memory addresses. Second, the binary data file lacked a header required by MATLAB. To overcome these two problems, Lt. Wintermyre created an application program which provided for the necessary alterations and conversions. The process for 200 kilopoints takes less than ten seconds.

The MATLAB manual states that "MATLAB is a high-performance interactive software package for scientific and engineering numeric computation. MATLAB integrates numerical analysis, matrix computation, signal processing, and graphics in an easy-to-use environment where problems and solutions are expressed just as they are written mathematically, without traditional programming" (MATLAB User's Manual 1). It is an application for the Macintosh which provides mathematical analysis as well as MATLAB specific "programming" for the user (though C and FORTRAN subroutines may be introduced).

There are two types of MATLAB programming files (M-files) a user can create. The first is called a script. A script is a series of equations and algorithms performed together. The user can specify and change all variables and equations from within the script. The other type a file a user can create is a function. A function allows the user to type in a discrete number of variable values; with these values, an algorithm is performed. With a script, the user can examine values for all variables present. With a function, the user can only examine the variable he specified to be returned.

Before any actual data was handled, simulated data was used to test how well the receiver components functioned. Figure 2 on 14-8 is a chart showing the scripts and functions that were created in conjunction with this project. When the simulated receiver components worked properly on the simulated data, actual data was introduced.

Figure 2: Scripts and Functions Created for Signal Processing

Scripts and Functions Created Prior to Data Acquisition

I. Main Scripts Created

1.WAVESTOTAL - Produce sine wave, composite sine wave, composite sine wave with noise, plot spectrum, filter out noise, plot roots and poles of a filter, frequency response of filter, comparison algorithms. Plots these operations.
2.AmplitudeMod - Simulates the modulation of a high frequency carrier wave and the demodulation of this wave using a half/rectifier and a low pass filter.
3.RFFilter - Simulates the work of an RFFilter on an airwaves environment. The filter pulls the desired channel from the airwaves. This script continues on to demodulate the channel, using a full rectifier and a lowpass Butterworth filter.
4.Sine Cosine AM Demodulator - Through an RFFilter, pulls a desired channel off the airwaves. Demodulates the signal through synchronous demodulation.
5.FULLAMRECEIVER -Simulates the work of an entire superhetrodyne AM receiver. See Figure 3 to see a copy of the script
6.FULLFMRECEIVER - Simulates the work of an entire superhetrodyne FM receiver See Figure 3 to see a copy of the script
7.Limiter - simulates the work of a bandpass filter on both frequency modulated and amplitude modulated waves
Various Butterworth filter plots/AM-FM experiments

II. Functions Created

1.AIRWAVES - creates an AM airwaves with up to five channels and random noise given carrier, signal, and sampling frequencies.
2.AMWAVE - creates a single AM wave given the carrier, signal, and sampling frequency
3.AMWAVEG - creates and graphs a single AM wave in the time and frequency domain.
4.DEMODAMCS - demodulates a signal employing the sine and cosine method given the function, carrier, signal, and sampling frequency
5.FTPLOT does a n-point Fourier transform given the function and the number of points (n).
6.FREQPLOT - plots the frequency response of a filter given the number of poles, the cutoff frequencies, and the number of points.
7.RFFILT - filters the airwaves and keeps a single channel given the function and the carrier, signal, and sampling frequencies,
8.FULLRECT - this function simulates the work of a full rectifier
9.HALFRECT - this function simulates the work of a half rectifier
10.DEMODFM - demodulates an FM wave - the fm wave is multiplied by a sine by a cosine - their sum is treated as a complex number - the complex number's angle is unwrapped - the derivative is taken of this answer- the derivative is divided by a constant(kf), which yields the modulation.
11.FMWAVE - creates an FM wave (created primarily to test other functions)

Scripts and Functions Created After Data Aquisition

I. Functions created for data handling

1.FCFIND - given data, this function allows the user to determine the carrier frequency of the AM or FM channel. This function can be used after demodulation to find the frequency of the modulating signal. The sampling frequency needs to be provided.
2.DEMODAMDAT - a spinoff of DEMODAMCS, this function heavily reduces the number of variables involved in the function processes. Many variables are used and cleared, while others are continually recycled. This allows the user to use a large number of sample points(25000-125000)
3.DEMODFMDAT - the counterpart to DEMODAMCSD for FM demodulation.

After introducing the data into the MATLAB environment, a need for modified functions was observed. These functions were designed to handle large numbers of samples. See Figure 2 for a chart of these functions as well.

A MEX function (an external function) called "playsound" allows the user to actually play the matrix data he introduces or derives with MATLAB. It provides an interface between MATLAB and the built-in speaker of the Macintosh. The function requires that the sample rate is known.

The scripts "FULLAMRECEIVER" and "FULLFMRECEIVER", created specifically for this project during the HSAP, were essential in completing the simulation. The scripts are found in Figure 3 starting on 14-10. There were also four functions developed during the HSAP essential to this project: DEMODAMCS, FCFIND, DEMODFM, and FREQPLOT. DEMODAMCS is found in the FULLAMRECEIVER script. The latter three have been included in Figure 4 on 14-13. Comments accompany the MATLAB scripts and functions. All MATLAB files were developed with MATLAB Version 3.5.

The most essential algorithms to this project were the demodulation algorithms. Both synchronous and asynchronous demodulation techniques were employed on the data. Asynchronous demodulation involves the use of an envelop detector. The IF signal is rectified and lowpass filtered. The DC (amplitude of the carrier wave) is removed by subtracting the mean of the lowpass filtered signal. In synchronous demodulation, the IF is split. One piece is multiplied by a cosine wave possessing the frequency of the IF. This *cosmult* signal is lowpass filtered. The other IF segment is multiplied by a negative sine wave possessing the frequency of the IF. This *sinmult* signal is lowpass filtered. For AM demodulation, the following operation is performed:

$$amplitude = \sqrt{sinmult^2 + cosmult^2}$$

This yields the amplitude of the AM signal. By removing the amplitude of the carrier (the DC), the modulation is found.

For FM demodulation, the *cosmult* signal is treated as the real part of a complex number, while the *sinmult* signal is treated as the imaginary part. The angle of the complex number is unwrapped, yielding the phase of the FM signal. The derivative of the phase yields the modulation times a modulation constant (Remember that in FM, the signal modulates the derivative of the angle). The modulation constant is divided out to yield the modulation. The functions designed for AM (synchronous and asynchronous) and FM demodulation are found in Figure 3, incorporated within the FULLAM and FULLFM receiver scripts.

Figure 3 Scripts FULLAMRECEIVER and FULLFMRECEIVER

```
%FULLAMRECEIVER simulates the work of an entire superhetrodyne AM receiver. First, the script
creates an airwaves consisting of five channels and random noise; these airwaves are "picked
up" by the "antenna" and viewed in the frequency domain. The recovered signal then goes
through the same processes that actual data would go through: RF amplification,channel
tuning, mixing, mixer filtering, IF amplification, demodulation, audio amplification, and
finally speaker output.

%A note on the mixer: Mixers are artifacts of actual receivers. They diminish cost as they
allow the demodulation process to work at only one frequency. This simulated receiver is not
constrained by cost. Thus, the mixer was left in for posterity.(The mixer filter causes a
shift in phase. If it is employed, the sine-cosine demodulation process will initially appear
to be incorrect).
%Created byWilliam Schatz July 20, 1993 on MATLAB Version 3.5

%Initial Variables
fs= 10000;                              %The sampling frequency
t= 0:1/fs:1-1/fs;                       %Time (as defined by the sampling frequency)
fcarrier = 3500;                       %Carrier wave frequency
fmod = 200                              %Modulation frequency (constant modulation used)

%Dummy signals (duma through dumh)
duma= 500;   dumb= 20;
dumc=1000;   dumd=300;
dume=1750;   dumf=150;
dumg=2500;   dumh=350;

%The antenna
fullrf=airwaves(fs,fcarrier,fmod,duma,dumb,dumc,dumd,dume,dumf,dumg,dumh)
%The airwaves function creates an "airwaves" complete with 5 channels and random noise

%The RF amplifier (Necessary in receivers to magnify signal reception)
ampcrf = 2;                   %Amplification factor
fullrfb= ampcrf*fullrf;          %Amplified Radio Frequency signal (RF)

%Channel Tuning (Tunes the Receiver into one channel)
tunedrf = rffilt(fullrfb,fcarrier,fmod,fs);    %Function "rffilt" tunes the receiver

%The Local Oscillator (Produces mixer oscillation)
freqif = 600;                          %Frequency at which all demodulation will occur.
foscil = fcarrier + freqif             %Frequency of the oscillation
oscillation = cos(2*pi*foscil*t);      %Oscillation wave

%The Mixer - Optional (see header)
intermf= oscillation.*tunedrf;              %Should always get an IF of freq 600
%Frequency plots (not included in script) will show four peaks (two represent the sum of
%fcarrier and foscil, and the other two represent the difference (which is always 600))

%The Mixer Filter - Optional (see note above)
%Actually, the receiver will work fine without this filter, and in some cases will work %even
better. However, it is easier to remove high frequencies now than during demod.

poles = 12;                            %Number of filter poles
cutoff = (fmod+freqif+50)/(fs/2);      %Cutoff frequency for the lowpass filter
[b,a] = butter(poles,cutoff);          %This function defines the roots/poles of filter
intermfb = filter(b,a,intermf);        %This function defines data passed through the filter

%The IF Amplifier  (Amplifies the IF so that it can be demodulated)
ampcif = 3;                          %Factor of amplification
intermfc= ampcif*intermfb;           %Amplified Intermediate Frequency(IF)

%Demodulation   (Separates the modulation from the carrier wave)
```

```
recovmod=demodamcs(intermfc,length(intermfc),fmod,freqif,fs);    %Demodamcs is a function
%created to demodulate AM waves through synchronous demodulation

function[magnitude] = demodamcs(y,points,fmod,freqif,fs)
t = 0:1/fs:1/fs*(points-1);                %Time - defined by sampling freq and points
sinif = y'.*(-2*sin(2*pi*freqif*t ));      %IF multiplied to sine wave
Wn= (fmod+5)/(fs/2);                       %Cutoff freq for filter
[b,a] = butter(12,Wn);                     %Defines filter with respect to a and b
lowpsif = filter(b,a,sinif);               %lowspif is the lowpass filtered data
cosif = y'.*(2*cos(2*pi*freqif*t));        %IF mult to cosine wave
lowpcif = filter(b,a,cosif);               %lowpcif is the lowpass filtered data
magnitude=sqrt(lowpsif.^2 + lowpcif.^2);  %Square Root-Sum of Squares of Lowpsif & Lowpcif
modulation = mean(magnitude) - magnitude; %Remove DC from the recovered Modulation
```

Asynchronous demodulation - AM demodulation can also be performed through asynchronous demodulation, which does not require knowledge of the carrier frequency. This method is commonly used in analog receivers. An envelope detector and a lowpass filter are the necessary digital components for simulation. An earlier version of FULLAMRECEIVER used asynchronous demodulation:

```
%Asynchronous demodulation
%The envolope detector
%Rectifier
for i= 1:1000               %This is a half rectifier
  if y(i)<=0, z(i)=0;end    %Negative function values become zero
  if y(i)>0, z(i)=y(i);end  %Positive values remain the same
end

%A full rectifier can also be used
%The Full Rectifier
for i= 1:1000   %Take the absolute value of the amplitude
  if y(i)<=0, v(i)=-y(i);   end
  if y(i)>0, v(i)=y(i);     end
end

%Butterworth Low-pass filter
n=8;                            %n-poles of the filter
Wn=60/500;                      %Cutoff frequency/Nyquist frequency
[b,a] = butter(n,Wn);           %Filter function
f= filter(b,a,z);               %The filtered recovered modulation

%Audio Amplifier (Amplifies modulation for sound conversion)
ampcaudio= 2;                     %Factor of amplification
recovmodb = ampcaudio*recovmod;     %amplified modulation

%Speaker
playsound(recovmodb,fs);        %Plays recovered modulation
bench = cos(2*pi*fmod*t);       %The ideal modulation signal
playsound(bench,fs);            %Play ideal modulation


%FULLFMRECEIVER-This script simulates the work of an entire FM receiver. A simulated FM wave
goes through the series of receiver components that actual FM data would go %hrough. This
includes several stages of amplification, a mixer. a limiter, a discriminator, and numerous
filters.

%This is a superhetrodyne FM receiver - it employs the use of a mixer to change the carrier
frequency. In actuality, this simulated receiver does not require a mixer. The simulated
receiver can demodulate waves at any intermediate %frequency. In actual receivers, a mixer is
used to cut down on cost - it is much esaier to design a receiver to work with only one IF.
Also, with simulated FM (no amplitude mod), a limiter is also unnecessary. Created by William
Schatz 8/3/93 on MATLAB Version 3.5
```

For convenience, those components identical in AM and FM receivers will not be redefined - however, they will be referred to.

```
%The FM wave itself
%The Variables and Function defining the modulation
fs= 100000;                             %The sampling frequency
points = 10000;                         %The number of points to b  sed in the script
t= 0:1/fs:(1/fs)*(points-1);            %The time as defined by th  sampling frequency
A= 1;                                   %Amplitude of the modulator
fm =  200;                              %Frequency of the modulator
mod = A*cos(2*pi*fm*t);                 %The modulation

%The Variables and Functions defining the modulated wave
r= 10;              %A constant (turns out to be the modulation index unless A isn't 1)
kf= r*2*pi*fm;      %Some constant defining the relationship between modulation and carrier
deltaf= kf*A;             %Deltaw is some variable necessary to define modulation index
m= deltaf/(2*pi*fm);      %m is the modulation index
fc = 30000;                                      %Frequency of the carrier wave
fmwave= cos(2*pi*fc*t + m*sin(2*pi*fm*t));       %The FM wave

%The RF amplifier
%The Local Oscillator
%The Mixer - Optional (see note above)
%The Mixer Filter - Optional (see note above)
%The IF Amplifier

%The Limiter (Removes amplitude modulation from FM wave)
VL = 1;                             %The high value limit
NVL = -1;                           %The low value limit
limit = zeros(intermfc);            %Create a vector with the same length as FM wave
Indexp = find(intermfc>0);        %Find index points of positive values
Indexn = find(intermfc<=0);       %Find index points of negative values
limit(Indexp) = VL*ones(Indexp);         %All positive points become VL
limit(Indexn) = NVL*ones(Indexn);        %All negative points become NVL
%Doing these operations yields a square wave. By bandpass filtering, the harmonics of %the
wave are removed, thereby smoothing the wave back into a sinusodial wave.
Wn = [(freqif-Mtot/2) (Mtot/2+freqif)]/(fs/2);        %Cutoff frequency for bandpass filter
[b,a]= butter(6,Wn);                                   %Poles of the filter
modfix = filtfilt(b,a,limit);  %Special filter function which produces no phase change.
%Effectively, the data is sent through the filter forwards and back

%The Discriminator -  Performs Synchronous demodulation for FM
%IF Multiplication with a Sine Wave and Filter
y=modfix;                                 %The pre-demodulated function
sinif = y.*(-2*sin(2*pi*freqif*t));       %IF multiplied to sine wave
lppoles=8;                              %The number of filter poles
Mtot = 2*(m+1)*fm;                      %Total bandwidth
Wn = (Mtot/2)/(fs/2);                     %Cutoff frequency for the lowpass filter
[b,a] = butter(lppoles,Wn);               %Defines filter
lowpsif = filter(b,a,sinif);              %Sinif is lowpass filtered
%IF Multiplication with a Cosine Wave and Filter
cosif = y.*(2*cos(2*pi*freqif*t));        %IF mult to cosine wave
lowpcif = filter(b,a,cosif);              %Cosif is lowpass filtered
modform = lowpcif + j*lowpsif ;         %Combine elements into complex signal
phase=unwrap(angle(modform));           %Get the phase of the modulated carrier by
                                        %unwrapping the angle of the complex number
deriv_phase= diff(phase)./diff(t);      %Derivative of the phase yields kf*modulation
modulation=deriv_phase/kf;              %Get the modulation
%Audio Amplifier
%Speaker
```

Figure 4 - Functions Created

```
function[fcarrier] = fcfind(x,fsampling, points)
%FCFIND(x,fsampling,points) is a function which determines the carrier frequency of wave
data. This is accomplished through a fourier transform of n points (if n is not defined, 2048
points is the default). The user should change the number of points when comparing decimated
and nondecimated data. Bill Schatz 8/4/93
if (nargin ==2)            %If the number of variables entered is two,
    points = 2048;    end  %it is assumed that 2048 points will be used
 X= fft(x,points);    %Fast Fourier transform of function(x) on n points.
k=0:points-1;         %Process to normalize the graph- not truly necessary, but helpful
W=(2*pi/points)*k;         %Omega equals 0 to 2pi
W= (W/pi) - 1;             %Omega goes from -1 to 1
viewer = abs(fftshift(X));           %Fast Fourier Transform shift on the points
index=find(viewer >= max(viewer)-1/20*(viewer)); %Find indices of values 19/20 or  %more of
the max - gives a more precise average and therefore a more accurate number
inpoints = abs(index - length(viewer)/2);        %Distance from the y-axis
inpoint = mean(inpoints) - 1;                    %Mean of the crucial index points
fcarrier = (inpoint/points) *fsampling;          %The carrier frequency


function[f] = rffilt(y,fc,fsignal,fs)
%RFFILT(y,fc,fsignal,fs) is a filter function employing a Butterworth
%filter. Given the data(y), the carrier frequency(fc), the modulation
%frequency(fsignal), and the sampling frequency (fs), RFFILT
%will remove the desired AM channel from the "airwaves". For
%0<fc<(fs/2-(fsignal+50)), the filter acts as a bandpass filter.
%For fc>=(fs/2-(fsignal+50)), the filter acts as a high pass filter
%with a stop frequency of fs/2-(fsignal+200). Bill Schatz 7/9/93
n=10;             %Number of poles is constant - 10
%The RF Filter
Wn= [(fc-(fsignal+50)) (fc+(fsignal+50))]/(fs/2);  %The size of the bandpass
[b,a]=butter(n,Wn);                               %The filter function
if fc>=fs/2-(fsignal+50)   %See description (this if loop specifies banpass vs highpass)
      [b,a] = butter(n,((fs/2)-(fsignal+200))/(fs/2),'high');  end
f=filter(b,a,y);       %The filtered data function


function[x,w,b,a] = freqplot(n,Wn,points)
%FREQPLOT(n,Wn,points) plots "points" points of the frequency response
%of a Butterworth filter given the filter's number of poles,
%the cutoff or bandpass frequencies, and the number of frequency response %points to plot. If
the number of points is not specified then
%512 points will be used. The freqplot is a semilog - y graph.
%Don't worry about sampling rates. The cutoff frequencies specify
%the Nyquist frequency(which is equivalent to one on the
%frequency axis).The frequency axis has been normalized from -1 to 1. Bill Schatz 7/7/93
if (nargin ==2)   points=512;  end
[b,a] = butter(n,Wn);      %This function defines the filter
[h,w] = freqz(b,a,points); %This function defines the freq response
x= abs(h);                 %Complex numbers included
w=w/pi;                    %Normalize the x-axis from -1 to 1
semilogy(w,x)              %Plot freq response (mag)
title('Frequency Response of the Filter')
xlabel('Frequency') ylabel('Magnitude')
```

Results

A first collection of data yielded three samples. Initially, it appeared that the MATLAB functions created only worked with simulated data. Data samples were demodulated and processed; however, when the data was applied to the "playsound" function, no sounds were heard. It also seemed possible that the data collections were made of "quiet airwaves"; that is, there were no voice or sound transmissions on the captured channel during the sampling interval. In addition, processing became nearly impossible because MATLAB's "Out of Memory" error message flashed frequently on the screen.

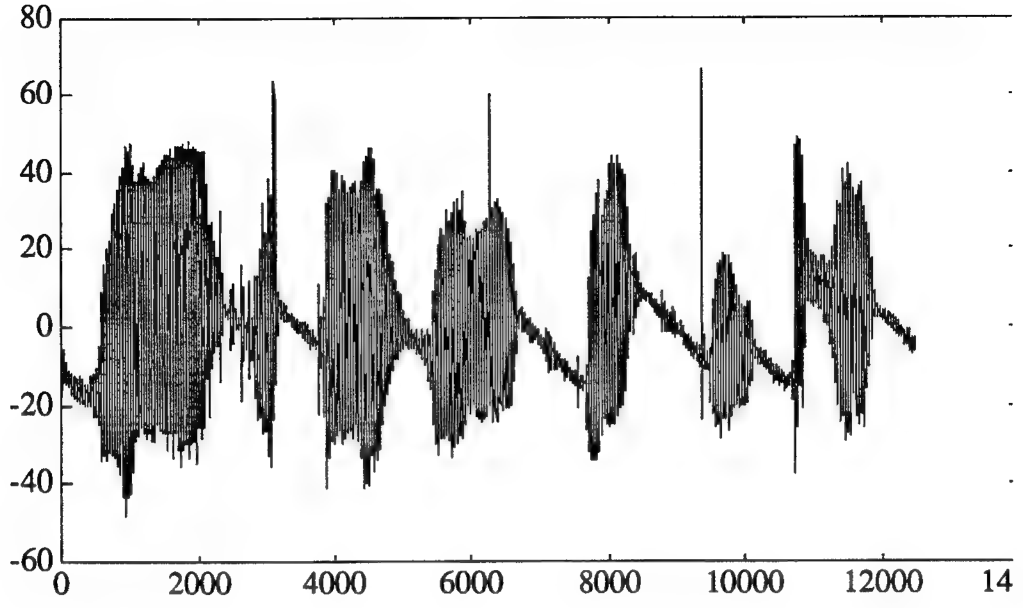
Handling the memory difficulties was relatively simple. The Macintosh IIx used in the investigation has 8 MB of RAM and an 80 MB hard disk. Activating virtual memory allows the user to expand the RAM to 13 MB. After activating virtual memory, the MATLAB application size was increased a substantial amount above the 1 MB suggested. MATLAB also possess a function called "pack", which rearranges variables in the workspace to create more room. This function was employed often.

The three raw samples themselves looked correct. The first sample was data from an AM wave emitted by a signal generator. The amplitude vs. time graph showed a well-formed modulation envelope. The second sample was data from an FM wave also emitted by the signal generator. Its time and frequency plots also appeared correct. The final sample was collected from the receiver. The A/D converter had been triggered during an air traffic control message; a female had delivered the message. This third sample's graphs also looked promising, especially in the frequency domain.

However, when the third sample was demodulated, the female voice was not heard. The data had been collected at a sampling frequency of 100 kHz. After investigation, it was learned that the maximum sampling frequency of the "playsound" function is 65 kHz. Lt. Wintermyre suggested that the data sample be decimated (i.e. the sampling rate of the data reduced). An existing MATLAB function was employed for this purpose.

The data sample was decimated in half. The new sampling frequency thus became one half the original, or 50 kHz. When this decimated data sample was applied to the "playsound" function, the female air traffic controller's voice could be clearly heard. The quality seemed higher than that of the receiver. See Figure 5 on 14-15.

Figure 5: Demodulated Signal - Woman's Voice: Air Traffic Control Message



Amplitude vs. Time

This is the amplitude versus time graph for the demodulated AM signal (female control tower operator voice). If the graph is examined, one sees two sharp single spikes at about 6000 and 9000. These are points of connection; the data had to be segmented for demodulation and decimation due to memory constraints. There is also a connection point at 3000, but it is not as readily apparent. At these points of connection there are clicking sounds because of these discontinuities in the recovered signal.

There are seven clusters of energy in this plot. Each cluster represents a word or sound. The message, when played using the "playsound" capability , states "Zero echo zulu squawk three five." The first two clusters of energy are probably that of "zero", while the remaining five clusters have a one to one correspondence with the remaining words.

Even though the demodulated signal shown in the graph has been decimated by a factor of 8 (this signal was originally decimated by 2, but was decimated more due to memory constraints) , the shape of the graph remains near identical to the shape of the pre-decimated plot. More importantly, the decimated signal has retained the audio clarity and sharpness of the original.

A second data collection was made yielding three new samples. Two samples were made of FM channels, while the third was made of an AM channel. One difficulty in obtaining FM signals is that only certain FM channels can be selected. The bandwidth of commercial FM stations is 200 kHz. Even with a new anti-aliasing bandpass filter with a bandpass width of 50 kHz (substituted for the original 20 kHz), only one fourth of a commercial FM station could be recovered. With a 50 kHz, bandwidth, commercial FM is barely recognizable. Thus, samples of narrowband FM, a airport weather station and a receiver "problem" channel, were taken. The antenna mounted to the roof has incredible range; a sample of a German news channel was this collection's AM data.

There was some difficulties demodulating the FM channels. A function employed in demodulation, the function "unwrap" used in unwrapping the complex numbers' angles, used too much memory for large samples to be processed. A new method was employed that essentially mimicked the function "unwrap"; it used less room and worked twice as fast. Lt. Wintermyre developed the mathematics behind the faster method and was instrumental in developing the MATLAB code. The function is found in Figure 6.

Figure 6 FM Demodulation Function

```
function[filtmod] = demodfmdat(y,points,fc,fm,fs)

%DEMODFMDAT(y,points,fc,fm,fs) is a function designed to demodulate
%(synchronous demodulation) a large number of FM signal samples. Many variables are cleared
%in the course of the function - MAKE SURE THAT YOU HAVE SOME
%BACKUP FOR THE  PREDEMODULATED DATA. See also the FULLFMRECEIVER script for another FM demod
%algorithm William Schatz 8/6/93

t = 0:1/fs:1/fs*(points-1);  %Time as defined by the sampling freq and points input
r= 5;                        %A relationship between the modulation and the mod constant
kf= r*2*pi*fm;               %The modulation constant
     clear r
m = 5;                       %The modulation index
sinif = y'.*(-2*sin(2*pi*fc*t));  %Multiply the data by a (-2)sine wave
cosif = y'.*(2*cos(2*pi*fc*t));   %Multiply the data by a (2)cosine wave
     clear y   clear fc
Mtot = 2*(m+1)*fm;              %Find total bandwidth
     clear m
[b,a] = butter(12,(Mtot/2)/(fs/2));  %The specifications of the lowpass filter
     clear Mtot
sinif = filter(b,a,sinif);          %Filter the sine multiplied signal
cosif = filter(b,a,cosif);          %Filter the cosine multiplied signal
     clear a  clear b
%This function know finds the derivative of the arctan of the expression u (defined as
%Q(t)/I(t). The derivative is equal to $\left(\frac{1}{1+u^2}\right)\frac{du}{dt}$ . The modulation is then found by
dividing the derivative by the modulation constant. The quality of the recovered mod can be
enhanced by a filter.

I =cosif;                           %Set I equal to lowpass cos mult term
Q= sinif;                           %Set Q equal to lowpass sin mult term
     clear cosif  clear sinif
A2 = I.^2 + Q.^2;                   %Define A (A is equal to u in the above equation)
dI =diff(I)./diff(t);               %Find the derivative of I
dQ = diff(Q)./diff(t);              %Find the derivative of Q
I = I(1:length(dI));                %Next three operations adjust lengths of vectors
Q= Q(1:length(dQ));
A2 = A2(1:length(dI));
I = (I.*dQ - Q.*dI)./A2;
```

```
clear dQ clear A2
I =I/kf;                            %Get the modulation
[b,a] = butter(8,(fm + 10)/(fs/2)); %Lowpass filter the recovered mod to improve quality
filtmod = filter(b,a,I);            %The filtered data
```

Summary of Results

Six samples were successfully made with the data collection apparatus. After applying these data samples to the original MATLAB scripts and functions, a need for modified algorithms and equations was observed. The entire full receiver routines had several unnecessary steps, while the functions developed could not handle large enough quantities of points. New scripts and functions were developed.

The clarity on the AM channels was quite high, and in some cases was higher than that of the receiver. However, the clarity on the FM channels was somewhat poorer than that of the actual receiver because the bandwidth was limited. The most the signal processing algorithms could handle at one time was 100,000 samples (a sampling frequency of 100 kHz would translate into a second's worth of data). Large samples had to be chopped, processed, and spliced. Hence, points of insertion in the data were accompanied with clicking.

The data itself had to be modified to be processed. Initially, a Macintosh application was created so that the data could be read as a MATLAB matrix. During actual processing, the data was chopped and decimated so that it could be demodulated and "played."

One thing not mentioned previously was the speed of the operations. The speed of some functions was exceptionally poor. A single demodulation and decimation process on 100000 sample points could take as long as 5 to 10 minutes. For an actual implementation of a computer simulated receiver, the speed of such device must be greatly enhanced.

Conclusions

The simulated receiver was a success, but not to the extent envisioned. The key to this investigation was that the proper conditions existed.

Conditions Necessary for Receiver Simulation:

- Antenna reception can be dealt with in its raw IF form
- This IF can be digitized and captured on the CompuAdd 325 and saved to a disk
- This data can be transferred from an IBM disk to a Macintosh disk
- The data can be introduced to the MATLAB environment
- The MATLAB environment can handle the introduction of outside data and process it

- The MATLAB environment and the Macintosh have enough memory capability to process a large number of sample points
- The data can be "played" through the speaker

All of these conditions were met; hence, the signal processing algorithms, functions, and scripts were successful.

What were the results envisioned for the receiver? The simulated receiver should be able to handle minutes worth of antenna reception and played it with exceptional clarity. Data cutting or decimation should not be necessary. It had also been desired that the receiver would work with relative speed.

These goals for the receiver were not met as hoped. The simulated receiver's success was hampered by the following limitations .

Specific Limitations Hampering the Simulated Receiver

- The analog to digital converter was of limited quality. The maximum sample rate attempted was 100 kHz. As a result, there was a great concern that there would be aliasing. To prevent aliasing, a bandpass filter was employed. This bandpass filter prevented samples of commercial FM channels from being practical. It also reduced quality on the wider-band AM channels.

- The equipment in the data collection apparatus is not perfectly calibrated, and thus some incongruities were observed.

- The biggest single limitation was memory size. The CompuAdd computer connected to the apparatus had difficulty obtaining more than 300 kilosamples in one sampling interval. Typically, data had to be cleared before a new sample could be attempted. The Macintosh IIx has only 13 MB of RAM with virtual memory active. Thus, the MATLAB application had to be allocated a deficient amount of memory. The algorithms could handle no more than 100 kilosamples at a time; if several large variables existed in the MATLAB workspace, the sample handling capability typically fell to under 50 kilosamples. Large data samples had to be segmented, processed separately and rejoined later. In addition, the smaller RAM size meant the operations would go relatively slow.

To make the receiver more successful, simple changes could be made. An "A/D" converter with higher sampling frequencies could be implemented. The equipment used in collecting antenna reception could be more precise. A computer with greater RAM and hard disk memory could be used in place of the Macintosh IIx. The Macintosh is merely a desktop; perhaps a workstation could be employed

However, the receiver will never approach real time operation unless drastic improvements are attempted.

The Future

The improvements in Digital Signal Processing (DSP) technology in recent years has been dramatic. DSP plug-in boards have been designed to install into computers for those machines working with complicated digital signals. Recent years have also seen the advent of DSP chips. In terms of software, many programs and applications, like MATLAB, have been designed to work in the realm of DSP. Computers serve to enhance the modularity of digital signals, thus allowing DSP operations to be quite flexible.

Improvements in computer hardware and algorithms could also serve to advance DSP. There has been growing interest in the last decade in the applications of Residue Number Systems (RNS) Arithmetic. RNS Arithmetic is based on the use of residues(remainders) to represent numbers. The residues can then be added, subtracted, and multiplied through modular arithmetic. RNS arithmetic is attractive because it is carry-free, allowing for great precision, speed, and minimal computer hardware. In particular, digital signal processing, with its large number of simpler operations, would benefit from this increased speed and simplicity. However, RNS has its drawbacks: division, magnitude comparison, and overflow detection require complicated algorithms and hardware implementation. (Jullien, ed.,et al 1-3).

A large number of mathematical papers have been written dealing with these problem topics. Large strides have been made towards making the use of RNS Arithmetic practical for computer hardware implementation. Entire computers have been built using RNS devices(Jullien, ed., et al 3). In fact, there is work being done for Rome Laboratory by contractors who are i nterfacing personal computers with RNS hardware designed to perform various DSP operations. There is alsoa project for RL where RNS operations are being emulated on software and implemented on a VAX 11/780. There exists a strong visible interest in the use of RNS Arithmetic for signal processing.

If there was a serious effort to implement a computer simulated radio receiver, a RNS based computer or processor could be developed to handle the DSP computations; the device would have to be specific to that one use only. RNS Arithmetic may be the key to making the simulated receiver closer to real time. If near real time were achieved, the computer simulated receiver could become widespread. The Macintosh IIx based computer simulated AM/FM receiver is merely a rapid prototype.

One possible application of the receiver not mentioned previously is as an educational tool. MATLAB scripts and functions are quite easy to alter and view. By changing values or components, the user can physically see how the output data is affected by employing MATLAB's graphing and playsound capabilities. The student can observe firsthand the steps necessary for a receiver to function. In addition, the simulated receiver employs enough digital signal processing theory and functions to give an observer or user a strong background in the area.

Works Consulted

Couch, Leon W., II. Digital and Analog Communication Systems. New York: Macmillan Publishing Company, 1990.

W. Kenneth Jenkins, ed., et al. Residue Number System Arithmetic: Modern Applications in Digital Signal Processing. New York: IEEE Press, 1986.

The MathWorks , Inc. MATLAB User's Guide. Natick, MA: The MathWorks, Inc., 1992. (unpublished)

Oppenheim, Alan V., and Ronald W. Schafer. Discrete-Time Signal Processing. Englewood Cliffs, NJ: Prentice Hall, 1989.

Oppenheim, Alan V., and Alan S. Willsky. Signals and Systems. Englewood Cliffs, NJ: Prentice Hall, 1983.

Technical Consultation

Lt. James Wintermyre -	MATLAB programming and troubleshooting, Modulation and Demodulation, Binary Data Conversion application, Digital Signals and DSP, General Consultation
Roland Holman-	Modulation and Demodulation, MATLAB, Receivers, Mixers, DSP, Data Collection
Michael Weir -	Data collection and Data Collection Apparatus, Receivers, Data conversion, DSP
Capt. Michael Meerschaert -	Receivers

Noise Reduction Improvement Testing: Comparing Old to New INTEL

Joseph C. Senus
High School Summer Apprentice

Final Report for:
Research and Development Laboratories
5800 Uplander Way
Culver City, Ca. 90230-6608

Noise Reduction Improvement Testing: Comparing Old INTEL to New INTEL

Joseph C. Senus
High School Summer Apprentice

Abstract

The INTEL algorithm modification developments under a project set up by a contractor were evaluated for their effectiveness when used as a preprocessor for either human or machine recognition. The plan presented by this contractor describes a series of tests that he ran on a certain INTEL we called Old INTEL. He then devised an INTEL that supposedly worked better than the Old INTEL we called this the New INTEL. Our project was to run both INTELs and compare the results to see if the New INTEL was actually better.

<u>EQUIPMENT USED</u>

<u>VRS-300:</u>

Voice Recognition System Model SYS300. The SYS300 provides voice data entry capability for a host Video Display Terminal. The host terminal retains its standard terminal keyboard and display capabilities. The SYS300 has the following capabilities:

a. A user-defined vocabulary of as many as 100 words and/or phrases
b. Trainable for any vocabulary in any spoken language
c. Reject threshold level control
d. User control of recognition parameters
e. Highly efficient real-time performance
f. 99+ percent accuracy, under test conditions
g. User programmable gain control

<u>Tascam DA-30 Digital Audio Tape Deck :</u>

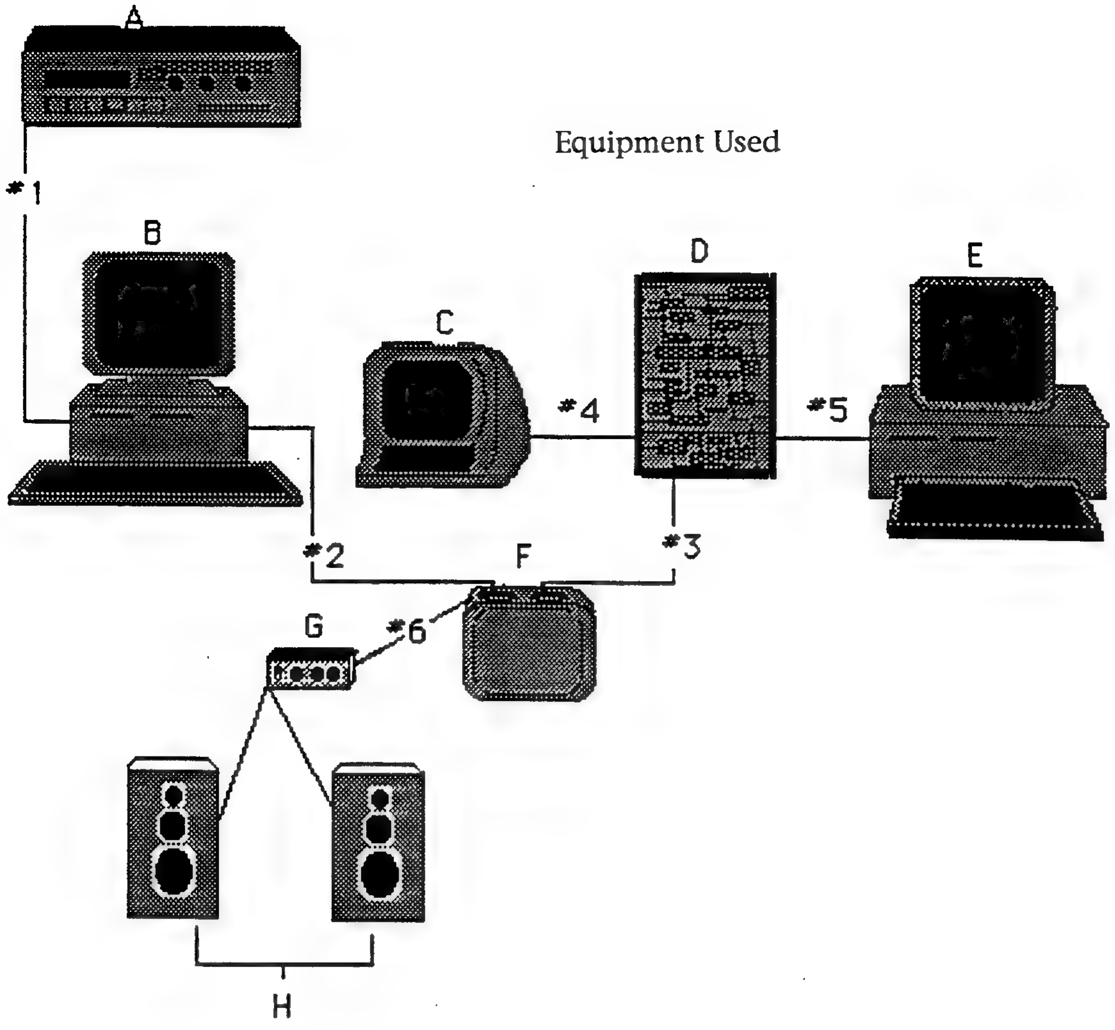
Plays and records digital audio

<u>Zenith PC with DSP-32C Coprocessor Board:</u>

The major hardware elements of the DSP(Digital Signal Processing Board)are:

DSP32C processor	The DSP-32C system's computational engine
Memory	Up to 256 Kbytes of program/data storage
Host port	The data port that interfaces to the PC and the DSP-32C
Analog I/O	Two channels of 16-bit data aquisition and digital to analog conversion
SCSI Interface	An industry standard disk drive/ peripheral interface
DSPnet Interface	High speed, 32-bit parallel I/O to other DSP cards

The DSP-32C is a single-board DSP system for IBM compatible computers. The DSP board adds up to 256 Kbytes of zero wait-state memory, SCSI and DSPnet ports and, two channels of professional-audio quality analog input/output.



Equipment Used

A. DAT player
B. Compaq with DSP board: converts analogue audio to digital audio and back to analogue
C. Adam III (Dumb terminal)
D.Voice Recognizing System (VRS sys300)
E. Zenith (host computer)
F. Audio Frequency Microvolter
G. Realistic SA-10 Solid State Stereo Amplifier
H. SONY speakers

*1. Line from DAT player to DSP board (analogue audio)
*2. Line from Compaq to AFM (analogue audio)
*3. Line from AFM to VRS (microphone)
*4. Line from Adam III to VRS (J2)
*5. Line from VRS to Zenith (host, J4)
*6. Line from AFM to amplifier

TEST PLAN

Demonstration of the INTEL Algorithm

The INTEL algorithm modifications developed under this project are to be evaluated for their effectiveness when used as a preprocessor for either human or machine recognition. The plan presented in this document describes a series of tests that will lead to both qualitative and quantitative evaluations.

Machine Recognition Tests

The SOW calls for a machine demonstration using the Interstate VRS-300 Recognizer. The demonstration procedure and recognition tests to be performed with this device are described in the paragraphs that follow, and the results are then analyzed.

The Experimental Database

The vocabulary contained in the audio data consists of the digits "zero" through "nine" spoken in English. The audio data is from one of the speakers of the NATO data base uttering five lists of single digits spoken in isolation, and each list contains 100 digits. The first list has each digit repeated ten times, the second list has the digits "one" through "zero" repeated in sequence ten times, and either of these lists can be used for training the VRS-300 . The remaining three lists have ten digits in a random sequence with each digit appearing ten times. Thus , there are ten utterances of each digit for training the VRS-300 and forty utterances of each digit are available in the recognition lists.

The audio data was supplied on magnetic tape with the noise free speech (NFS) on one track and wide band random noise on the second track. The volume of the utterances in each list was founds to vary as much as 4 dB. The volume of the noise track was found to have a variability of less than 2 dB. The signals on the two tracks were mixed to create speech plus noise (SPN) signals that have the following signal-to- noise ratios (SNR):

1. SPN1: SNR between 12 dB and 16 dB, moderately noisy;
2. SPN2: SNR between +4 dB and +8 dB, noisy;
3. SPN3: SNR between -7 dB and +3 dB, very noisy.

There will be four categories of audio data in the database, with each category comprising both the NFS and the SPN material. The categories are: [BASE], [noINTEL], [INTEL*2], and [INTEL*4] .

INTEL Processing of the Audio Data

The INTEL parameters for processing the various categories are as follows:

Sampling rate:	16 kHz
Window size	2048 Samples
Overlap and Weighting	50% with triangle function
Noise estimate update	0.8 seconds time constraint
Noise threshold	High
AGC	Peak

For each list that is processed , the input to INTEL is recorded simultaneously with the INTEL output on separate tracks of a cassette tape, such that when reproduced the input appears on the left channel and the INTEL output on the right channel.

Training Patterns for the VRS-300

The VRS-300 is a speaker dependent recognition device that must first be trained with the vocabulary that it is to recognizes uttered by the speaker that provides the utterances for the recognition task. Furthermore, the VRS-300 requires speech at a high SNR. for the training task. Consequently, the recognition tests will be performed with training patterns from the different categories of audio data that are expected to have a sufficient SNR. A training pattern (TP) is either of the training list utterances, form the NFS or SPN material, that is processed by INTEL. The training patterns are;

1. TP{ NFS[BASE] } = noise free, original:
2. TP{ NFS[noINTEl] } = noise free, "bypass";
3. TP{ NFS[INTEL*2] } = noise free, INTEL*2 process ;
4. TP{ NFS[INTEL*4] } = noise free, INTEL*4 process;
5. TP{ NFS[INTEL*4] } = Noisy (-7 dB <SNR< 3dB), INTEL*4 process;

Recognition Patterns for the VRS-300

For each training pattern that is loaded into VRS-300, and described above, different recognition patterns will be used for the recognition task. a recognition pattern RP) is comprised of the four recognition lists, of 100 single digits each, that are either unprocessed or processed by INTEL., The RP's are;

1. RP{ NFS[BASE] } = noise free, original ;
2. RP{ NFS[noINTEL] } = noise free, "bypass" ;
3. RP{ NFS[INTEL*4] } = noise free, INTEL*4 process;
4. RP{ SPN[BASE] } = Noisy (-7 dB <SNR< 3dB), original;
5. RP{ SPN[noINTEL] } = Noisy (-7 dB <SNR< 3dB), bypass;
6. RP{ SPN[INTEL*2] } = Noisy (12 dB <SNR<16dB), INTEL*2 process;
7. RP{ SPN[INTEL*4] } = Noisy (-7 dB <SNR< 3dB), INTEL*4 process

Training/ Recognition Tests Matrix

The matrix of recognition versus training patterns shown on the next page gives the purpose of the recognition tests.

TRAINING PATTERNS	NFS BASE	NFS noINTEL	NFS INTEL*2	NFS INTEL*4	NFS INTEL*4
NFS [BASE]	(1)	(2)			
NFS [noINTEL]	(2)		(2 3)	(2 3)	(3)
NFS [INTEL*4]	(2 3)	(2 3)	(3)	(3)	(3 4)
SPN [BASE]	(1)				
SPN [noINTEL]		(2 4)			
SPN [INTEL*2]		(3 4)	(3 4)	(3 4)	(3 4)
SPN [INTEL*4]		(3 4)	(3 4)	(3 4)	(3 4)

Each recognition test has one or more purposes as indicated by the parenthesized numbers as follows:

(1) This test checks the operation of the VRS-300 or provides a reference recognition score for the NFS and the SPN case.

(2) This test reveals if the INTEL program framework introduces any distortions in the output signal that degrades the VRS's performance.

(3) This test reveals if the INTEL enhancement process introduces any distortion in the output signal that degrades the VRS's performance.

(4) This test reveals whether the INTEL algorithm's enhancement improves the performance of the VRS-300.

Setup of the VRS-300 Demonstration

Audio input, monitored through a loudspeaker, is applied to the VRS-300 at its microphone input jack for both the training and recognition tasks. The signal must also be monitored to ensure that the proper level is maintained and that the signal is not degraded by distortion. This is done with a two channel oscilloscope where one channel monitors the output of the cassette playback device which is fed through an attenuator to the VRS-300 microphone input jack. The second channel of the oscilloscope monitors the output of the VRS-300 internal microphone preamp for distortion or limiting.

The digital port of the VRS-300 is connected to an Zenith computer that controls the VRS-300 operation and displays the recognition results on the CRT screen. The results are noted down for subsequent analysis.

VALIDATING THE FPASP4 SOFTWARE DEVELOPMENT ENVIRONMENT

Eric Hayduk

Final Report for:

AFOSR Summer Research Program

Rome Laboratory

Sponsored by:

Air Force Office of Scientific Research

Bolling Air Force Base, Washington, D. C.

August 1993

VALIDATING THE FPASP4 SOFTWARE DEVELOPMENT ENVIRONMENT

Eric Hayduk

Abstract

The validation of the FPASP4 (Floating Point Application Specific Processor v 4.0) software development environment was initiated. To validate the FPASP4, basic language structures were written in C, compiled in MIPS R3000 assembly language, translated into FPASP4 assembly language, then evaluated by using a simulation of the FPASP4 chip. The results show that many of the basic language structures tested translate and function properly. Results also show that some of the language structures tested were either mistranslated or did not function properly. These language structures point to problems with both the assembler and the translator. In some cases, the MIPS assembly language instructions were mistranslated or were not supported by the translator, while the assembler did not accept some of the required MIPS assembler directives. Tested structures are discussed along with problems, corrections, and future steps to complete validation.

VALIDATING THE FPASP4 SOFTWARE DEVELOPMENT ENVIRONMENT

Eric Hayduk

Introduction

The Wafer Scale Vector Processor (WSVP) and FPASP4 software environment are being developed by Air Force Rome Laboratory. WSVPs are constructed using FPASP4s. The WSVP is an adaptable and highly efficient processor that offers many advantages in the field of signal processing. (Signal processing focuses on the detection and tracking of both ground and airborne targets.) The main advantage of the WSVP is its ability to handle floating point calculations quickly and efficiently. Other advantages include small size, low weight, and low power consumption. These advantages are a result of chip stacking, a new memory packaging approach, a streamlined architecture design and hybrid wafer scale integration. Chip stacking involves thinning down the backs of several chips and stacking them on top of one another to save space. Hybrid wafer scale integration is the process of producing chips and then placing them on a wafer, edge to edge, with minimal spacing. The improvements in memory packaging allow for a greater reduction in the size of the WSVP and better memory organization. The FPASP4 software environment incorporates RISC-like instructions that include loads, stores, and simple arithmetic operations. RISC (Reduced Instruction Set Computer) architecture utilizes simpler instructions to increase performance. This occurs because compilers (which optimize programming code) almost always use simpler assembly language instructions rather than more complex assembly language instructions.

The WSVP can be used in many applications relating to signal processing. The processor can easily be used in both ground and airborne radar systems because of its efficiency, compact size, and low power consumption. The WSVP can be especially useful in designing satellites because satellites have demanding size, weight, and power requirements.

This report details the testing of the FPASP4 software development environment and the MIPS R3000 to FPASP4 assembly language translator. The reason for testing is to confirm the accuracy and completeness of the FPASP4 assembly language, assembler, and the translator and document errors in the FPASP4 assembly language, assembler, and the translator so they may be corrected.

Methodology and Procedures

The first step in validating the FPASP4 assembly language was writing basic language structures in C, which is a general purpose, procedure oriented programming language. These structures were divided into three major categories: control, operations, and other structures such as subroutines, functions, and arrays. Control structures include loops (while, do while, if-else, for, switch) and comparative structures (==, +=, >=, <=, >, <, ++, --, !=). Operation structures include simple math applications, such as adding two integers, subtracting one integer from another, multiplying two integers and dividing two integers. Math applications involving single and double precision floating point numbers were also included in operation structures. Other structures also tested include arrays, subroutines called by value, subroutines called by reference, functions, and recursive functions. Again, each was tested using integers, single and double precision floating point arithmetic. The following are examples of some of the language structures tested:

```
#define LIMIT 20

main()
{
int counter, value = 0;
for (counter = 0; counter <= LIMIT; counter++)
 { value++ }
}
```

Example 1: A for loop

```
main()
{
float x = 125.8934, y = 127.6431, z = 0;

z = x + y;
}
```

Example 2: Addition of two floating point numbers

```
#define N_MAX 5

main()
{
int number = 11, values[ N_MAX ];
values[0] = number;
```

Example 3: An array

```
values[1] = number;
values[2] = number;
values[3] = number;
values[4] = number;
}
```

Example 3: Continued

After verifying that the basic C structures compiled and functioned properly, the next step was to produce MIPS R3000 assembly language code. This was done by compiling the C source code on a MIPS R3000 workstation with an option that produced MIPS R3000 assembly language code. The following is an example of MIPS R3000 assembly language code.

```
          .verstamp  2 20
          .text
          .align     2
          .file      2 "for_loop.c"
          .globl     main
          .loc  2 8
 #  8 {
          .ent  main 2
main:
          .option    O1
          subu  $sp, 8
          .frame     $sp, 8, $31
          .loc  2 8
          .loc  2 12
 #  9
 # 10 int counter;
 # 11
 # 12 for (counter = 0; counter <= LIMIT; counter++)
          sw    $0, 4($sp)
          .loc  2 13
 # 13 {}
$32:
          lw    $14, 4($sp)
          addu  $15, $14, 1
          sw    $15, 4($sp)
          ble   $15, 20, $32
          .loc  2 15
 # 14
 # 15 }
          move  $2, $0
          .livereg   0x2000FF0E,0x00000FFF
          addu  $sp, 8
         j      $31
          .end  main
```

Example 4: MIPS R3000 version of a for loop

The MIPS R3000 assembly language code was then translated to FPASP4 assembly language code using the MIPS R3000 to FPASP4 translator. Example 5 is an example of FPASP4 assembly language code:

```
          .include "startup.a"

;              .verstamp  2 20
               .text
               .align     2
;              .file      2 "for_loop.c"
;              .globl     main
;              .loc  2 8
;   8    {
;              .ent  main 2
main:
;              .option    01
               subui      sp, sp, #8
;              .frame     $sp, 8, $31
;              .loc  2 8
;              .loc  2 12
;   9
;  10    int counter;
;  11
;  12    for (counter = 0; counter <= LIMIT; counter++)
               sw    #4[sp], lrz
;              .loc  2 13
;  13    {}
$32:
               lw    $14, #4[sp]
               addui      $15, $14, #1
               sw    #4[sp], $15
               cmpi  $15, #20
               br    UILE, #$32
               nop
;              .loc  2 15
;  14
;  15    }
               mov   $2, lrz
;              .livereg   0x2000FF0E,0x00000FFF
               addui      sp, sp, #8
               jmp_r      $31, #0
              nop
;              .end  main
```
Example 5: FPASP4 Assembly Language version of a for loop

After the FPASP4 assembly code was produced, the program being tested was executed on a simulation of the FPASP4 chip. The simulation was created by using another language called VHDL (VHSIC Hardware Description Language) which approximates the functions of hardware, in this case, the FPASP4 chip. Output files were evaluated to confirm that the program being tested functioned properly.

Results

The C language structures that correctly functioned did so because the structures correctly translated and assembled in FPASP4 assembly language. The following is a list of tested structures that functioned correctly:

1. *While loop*
2. *Do while loop*
3. *If else loop*
4. *For loop*
5. *Various integer compares (==, +=, >=, <=, >, <, ++, --, !=)*
6. *Arrays*
7. *Integer addition*
8. *Integer subtraction*
9. *Integer multiplication*
10. *Integer division*
11. *Subroutines called by reference*

The C language structures that did not function correctly did so because each language structure either mistranslated, was not supported by the translator, or the assembler did not accept some of the required MIPS assembler directives. The following is a list of C language structures that did not function properly and why they did not function:

1. *Switch loop*
   The switch loop did not function because the `sll` statement was not supported by the translator, and two statements were not supported by the assembler (.word, .rdata).
2. *Various single and double precision compares (==, +=, >=, <=, >, <, ++, --, !=)*
   The various compares do not work because the translator does not support floating point instructions (cvt.d.s, c.eq.d, bclf, li.d, c.le.d), the mistranslation of li.s and the failure of the assembler to recognize MIPS R3000 $f4-$f10 and $f16-$f18 registers.
3. Single and double precision floating point operations
   The single and double precision floating point operations do not work because the translator does not support floating point instructions (cvt.d.s, cvt.s.d, li.d) and because the assembler fails to recognize the MIPS R3000 $f4-$f10 and $f16-$f18 registers.
4. *External variables*
   External variables do not work because an error occurs in translation whenever external variables are used. The error causes the language structure to malfunction.
5. *The #include <stdio.h> statement*
   This statement does not work because the assembler does not support the equivalent MIPS R3000 statements (.extern _iob 0, .extern _bufendtab 0)
6. *Recursive functions*
   Recursive functions translate and assemble correctly, but will not simulate.

All of the incorrectly functioning language structures point to problems in the either the translator or the FPASP4 assembly language. One of the most common problems is MIPS R3000 assembly language statements that are not supported by MIPS R3000 to FPASP4 translator. Unsupported statements result in translator syntax errors and are translated as blank lines (see example 6) in the resulting FPASP4 code.

```
        MIPS R3000                              FPASP4
#  8  if (x == y)                       ;  8  if (x == y)
     c.eq.d      $f4, $f6
     bclf        $32
     .loc        2 9                    ;    .loc      2 9
 # 9      flag = 1;                     ;  9     flag = 1;
```

Example 6: Example of translation of unsupported statements

The following is a list of unsupported MIPS R3000 statements:

1. cvt.d.s:	This statement translates as a blank line.
2. c.lt.d:	This statement translates as a blank line.
3. cvt.s.d:	This statement translates as a blank line.
4. c.eq.d:	This statement translates as a blank line.
5. bclf:	This statement translates as a blank line.
6. bclt:	This statement translates as a blank line.
7. trunc.w.s:	This statement translates as a blank line.
8. li.d:	This statement translates as a blank line.
9. li.s:	This statement mistranslates.
10. div:	This statement translates as a blank line.
11. sll:	Only part of the statement translates. i.e. `sll $15, $15, 2` becomes `$15, $15, 2`
12. c.le.d:	This statement translates as a blank line.
13. mfcl:	This statement translates as a blank line.
14. l.d:	This statement translates as a blank line.

Some statements are supported by the MIPS R3000 to FPASP4 translator but are not accepted by the FPASP4 assembler. These statements result in syntax errors when the tested program is assembled. The following list documents MIPS R3000 assembly language statements that translate adequately but are not accepted by the FPASP4 assembler:

1. .extern _iob 0
2. .extern _bufendtab 0
3. .rdata
4. .word
5. .sdata
6. .align 2
7. . float
8. 1.0000000000000000e+00
9. $f4-$f10 and $f16-$f18 registers

There are also other problems with the assembler and the translator. Registers that the MIPS R3000 assembly language utilizes for temporary storage, $f4-$f10 and $f16-$f18, are not recognized by the assembler and are not placed on their respective busses. In other programs variables become undefined due to mistranslation. For example this happened in the FPASP4 version of a program that called a subroutine by value. Finally some programs would translate and assemble adequately, but would not simulate.

Future stages

There are several future steps that should be done to complete validation. First, more basic language structures could be written and analyzed. C language structures that should be supported by the translator and the assembler include printf, scanf, and strings. After validation, the errors in both the translator and the FPASP4 assembler should be corrected in order to increase the their effectiveness.

Conclusion

The WSVP is an effective and efficient processor that can handle floating point calculations quickly and efficiently. The FPASP4 assembly language and the MIPS to FPASP4 translator support many of the basic language structures tested although several are either mistranslated or are not supported by the translator and the assembler. Correcting these errors and completing validation are essential to the completion of this project.

REAL TIME 12 BIT ANALOG TO DIGITAL COMPUTER INTERFACE

Christopher M. Vaill

Final Report for:
AFOSR Summer Research Program
Rome Laboratory

Sponsored by:
Air Force Office of Scientific Research
Bolling Air Force Base, Washington, D.C.

August 1993

REAL TIME 12 BIT ANALOG TO DIGITAL COMPUTER INTERFACE

Christopher M. Vaill

Abstract

Two prototype printed circuit boards were designed and constructed for use in the computer interface to a microwave life test system at Rome Laboratory. Computer aided design (CAD) tools were used for both schematic circuit design and printed circuit design. The prototype circuit boards were created with a computer controlled milling machine. Upon testing, both the A/D converter and the power conditioner board had minor errors which were corrected for the final circuits. In addition, minor output errors in both CAD programs were overcome with a bit of troubleshooting and three post processing programs written in the C language.

REAL TIME 12 BIT ANALOG TO DIGITAL COMPUTER INTERFACE

Christopher M. Vaill

Introduction

The main thrust of my high school apprenticeship program at Rome Laboratory concentrated on design and construction of two printed circuit boards for an automated microwave life test system. Because all multitask computer systems (even those as small as PCs) devote considerable amounts of time to tasks other than running the user's test program, they are not directly compatible with real-time data acquisition. Outside hardware is required to buffer real time data and pass it to the computer system where final data reduction takes place. The power conditioner printed circuit board has voltage regulators and filters for both the device under test and the analog to digital converter. The analog to digital converter printed circuit board has the A/D converter as well as all control logic to interface with the Hewlett Packard model 380 computer. Computer aided design (CAD) systems were used extensively.

Discussion of Problem

Because the preliminary electronic circuit design already existed, my initial tasks were:

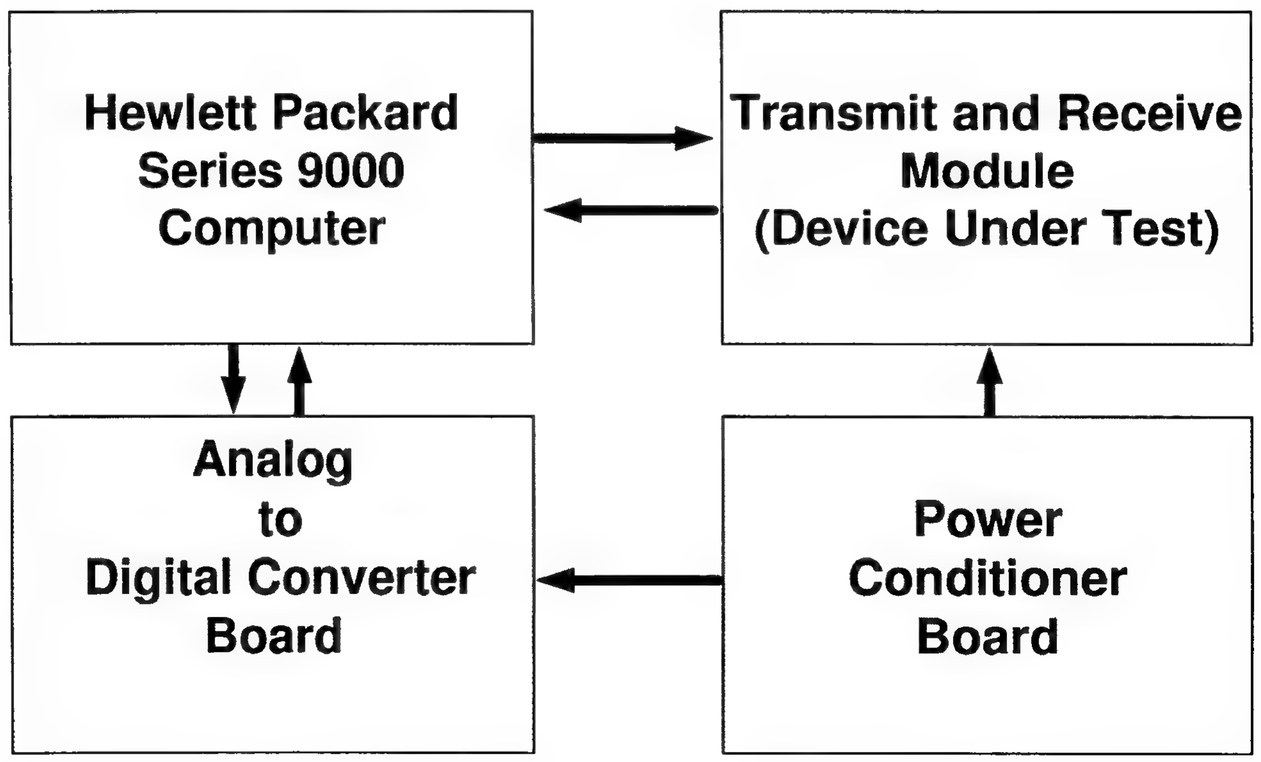
1. Finalize the schematic diagram for the printed circuits on a CAD system
2. Load the CAD-PC (printed circuit) software on an IBM 486 personal computer
3. Create a CAD-PC file for the analog to digital converter board
4. Create a CAD-PC file for the power conditioner board
5. Use CAD-PC to fabricate the A/D converter board
6. Use CAD-PC to fabricate the power conditioner board
7. Put the components on the A/D board
8. Put the components on the power board
9. Test the A/D board and the power conditioner board

Two additional tasks arose during the course of the project:

10. Troubleshoot why the CAD (schematic generation) software would not work properly with E-size plots (42" by 32")

11. Troubleshoot why the CAD-PC (printed circuit) software would not put circuits out to the Hewlett Packard plotter properly

Microwave Life Test System


Hewlett Packard Series 9000 Computer
Transmit and Receive Module (Device Under Test)
Analog to Digital Converter Board
Power Conditioner Board

Methodology

The system block diagram indicates major components in the life test system. The shaded components were worked on during this apprenticeship. One Hewlett Packard computer controls up to 40 transmit and receive modules. Each transmit and receive module has a power conditioner board and an analog to digital converter board. The analog to digital board is capable of digitizing up to 16 different power conditioner voltages or currents and sending this information to the HP computer. The power conditioner provides the voltages necessary to run the device under test. Initially both the power conditioner and monitoring A/D converter were to be on one printed circuit board. However, the computer controlled milling machine which is used to fabricate prototype boards was near its size limit and noise isolation is easier with two circuit boards, so the decision was made to use two boards.

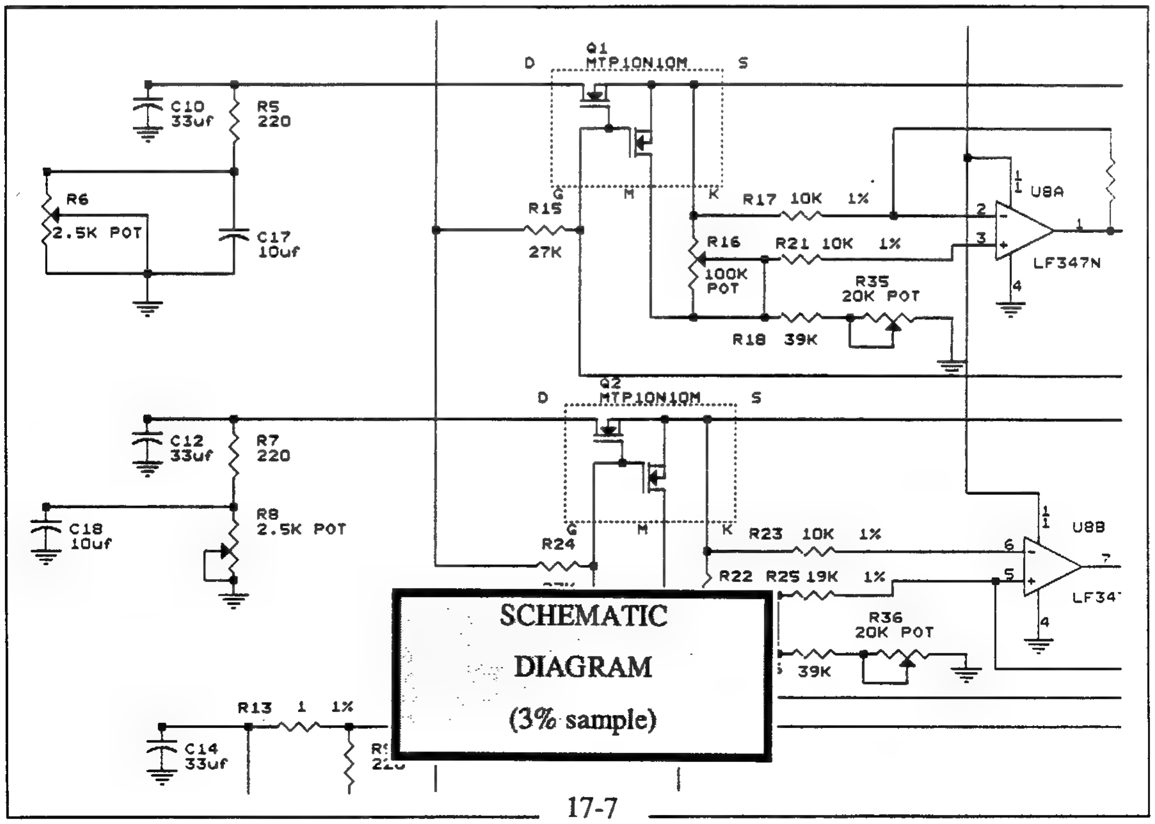
The CAD software used to create schematic diagrams and much of the schematic diagram itself were already loaded on the IBM personal computer. Up to this point the schematic had only been viewed on the computer monitor. My first task was to add a few final parts and connections to the the schematic diagram and plot out a hardcopy. This plot would then be used to assist in creating the printed circuit board layouts. The schematic changes went well enough. To make the paper plot, a plot file was created on the personal computer. This was then sent via network to the Hewlett Packard 9000 series computer. The HP 9000 had a Draft Master RX plotter connected to it via HPIB (Hewlett Packard Interface Buss). A simple copy of the plot file from the HP to the plotter device number would create the plot. However, what was created was the left two thirds of the schematic diagram and a few short lines off the left side of the paper rather than the right hand one third of the diagram. After a couple of more attempts with the engineer who was my lab focal point, we enlisted the assistance of the engineer who set up the Draft Master RX a couple of years ago. The first thing he had me do was construct a 50 foot serial cable to go between the personal computer and the plotter directly. This took about two hours. The thought was that the network or a filter in the HP 9000 might be losing data when some buffer was full and the handshakes did not work correctly. The direct connection ended up with similar results as the network. However, the direct connection made it easy and fast to try sample plots and see where the system failed. After another two hours we had a couple of smaller schematics which indicated the trouble only occurred if parts were more than 30 inches to the right on the plot. Because Hewlett Packard had the good sense to use only standard ASCII characters in their plot files, we could then look at the file with a standard text editor, well almost (the data was all in one 10,000 character line). At this time a C program was written:

```
#include <stdio.h>  /* addcrlf1.c to add CRLF after any ; */
 main(argc, argv)
 int argc;
 char *argv[];
{ int a, b, c, ch;
 unsigned int i;
 FILE *fpr, *fpw, *fopen();
 fpr = fopen(argv[1], ''rt'');
 fpw = fopen(''fixed'', ''wt'');
 while((ch = getc(fpr)) != EOF){
 if(ch==';'){ putc(ch, fpw); fprintf(fpw,''\n'');
                               //put CRLF to output   file }
 else { putc(ch, fpw); //end of else
 }//end of while
 fclose(fpr); fclose(fpw);
}//end of main
```

which processes any file listed as the first argument on the command line and gives an output file named "fixed" which has a carriage return and line feed after each and every semicolon in the file. This file can be readily examined with an ordinary text editor. This program took "us" about one half hour to create (I do not program in C at the moment). At the end of the day the best guess was that software vendor had upgraded compilers for this version of the schematic program (version 4.10) and it defaulted to signed integers rather than unsigned integers. Every time a location number got greater than 32.768 (or 32,768 thousandths of an inch on the plot) the compiler interpreted this as two's complement negative number. Hence, the short lines off the left of the plot paper (the plotter quit at the hard clip limit about one quarter inch to the left of x=zero). Another C program was created the next morning in an attempt to post process the plot file:

```
#include <stdio.h>  /* schfix.c   E size plot files*/
 main(argc, argv)
 int argc;
 char *argv[];
{
 int a, b, c, ch;
 unsigned int i;
unsigned char numstring[8]='''';
FILE *fpr, *fpw, *fopen();
fpr = fopen(argv[1], ``rt'');
fpw = fopen(``fixed'', ``wt'');
 while((ch = getc(fpr))  != EOF){
   if(ch=='-'){      fscanf(fpr, ``%u'', &i);
                 //get absval of neg number into i
 if(i>16000)i=65536-i;
 if(i<16001) putc(ch, fpw);
     fprintf(stdout,''%u'', i);
fprintf(fpw,''%u'',i);  //put to real file
               } // end of if <16001
      else             {
      putc(ch, fpw);   }   //end of else
                             }//end of while
fclose(fpr);
fclose(fpw);
} //end of main
```

This program reads in any file supplied as the first command line argument and changes all negative numbers over 16,000 to two's complement positive numbers and puts them out in a file named "fixed". The fix actually worked and on the third day I got back to designing printed circuits for the power conditioner and A/D converter. A small (3%) sample of corrected schematic output is shown below:



The creation of a printed circuit board layout with the CAD program requires several steps. Although the program allows the user much freedom in the way of organization, the manual advises the creation of a parts library first, from which common components may be imported. The FLASH directory is the main directory in which the CAD software is installed, so I created a subdirectory LIB in which to store parts. Each of the parts is created in the same manner as a full circuit board. This allows for the importing of smaller circuits into larger ones, in this way large parts are often made up of other, smaller ones. For example: I created parts for several different voltage regulator packages, and each of them needed a heat sink, so I used a single heat sink part and imported it into each voltage regulator package design.

Each board layout consists basically of three items: pads (holes surrounded by conductor material), conductor routes (circuit board copper conductor areas isolated by thin lines where the conductor has been removed), and supporting lines (lines which visually assist the operator and have no effect on the actual circuit, such as the physical outline of a part). To begin creation of a part, measurements must be taken of the component's physical dimensions. Not only is this necessary for correct hole placement, but also for the supporting lines, which must be accurate enough to let the operator prevent two (or more) parts from occupying the same space.

The first step in the CAD printed circuit generation program is to set conditions. The conditions are groups of values that correspond to photo data, tool sizes, and pad values. In the photo data list, the operator specifies line widths and pad apertures. (The pad aperture values include only the conductor surrounding the hole and not the hole itself. The aperture values and hole values are combined to create pad values in a separate menu.) These are just arbitrary values that the operator selects; they determine the actual widths of conductors and pads on the final circuit board. I used two line widths on the power supply board, .065 inches and .165 inches, because those were two values that would fit evenly on the .05 inch grid (with .035 inch milled-out lines between them). The pad apertures I used included those same two widths, as well as a square pad for isolating the pegs of heat sinks and a .095 inch pad created especially for a pin on one connector that required plenty of conductor around it and was not restricted by its proximity to other elements.

Later, when a line must be drawn on a circuit layout, rather than specifying the actual width, one is chosen from the list of available widths. This is a great convenience if all the lines of one width must be changed. Rather than manually going through an entire layout to change each line individually, a single entry in the conditions menu can be altered, which immediately affects all the lines that were created with that entry specified. Pad apertures, as well as all the other conditions, can be manipulated in the same way.

The tool size conditions are used to define the sizes of bits the machine uses for milling, drilling, and hatching (removal of excess copper between conductors). I found the best way to set these was by going through the actual bits we had available and entering their sizes. This eliminated the possibility of creating lines in the design that would require fabrication by a tool we did not have.

Finally, the pad values had to be created. The pad conditions require no new information, as each pad is made up of a hole size and a pad aperture, both of which having been previously defined in the drilling conditions and photo data menus. I used the .065 inch aperture with a .039 inch (1.0 mm) hole for most pads, and with a .055 inch (1.4 mm) hole for several pads requiring larger holes, including a voltage regulator and a 25-pin connector. I used the .165 inch aperture with .063 inch (1.6 mm) holes for the fuse holders and for a raw voltage input connector, and with .079 inch (2.0 mm) holes for some of the thicker jumper wires. My last pad was made up of the .165 inch square aperture with a .118 inch (3.0 mm) hole, into which the heat sink pegs were inserted.

Next, the dimensions are used to place the pads for the part. The choice of pads is very important. First, the hole must be the right size. If the hole is too small the pin will not fit through it, and if the hole is too large it makes the pin difficult to solder. The area of conductor around the hole must be as large as possible to accommodate the solder, but not so large as to overlap on other pads nearby. This is especially important with most IC packages, since their pins are just a tenth of an inch apart. For these I used a pad that was just the right size, taking into account the width of the milling lines, so that milling lines of adjacent pads were tangent to each other (refer to the top view of the A/D converter board on page 17-13). I also found that the pad can be too large. On some of the ground connections I used just a hole in the middle of a large area of copper. When I tried to solder these, the highly conductive copper distributed away much of the heat from the iron, making it difficult to get the immediate area around the hole hot enough to make the solder flow.

Adding support lines is also an important step. Although these lines are invisible to the milling machine, they are essential for designing the layout. Besides showing the operator the sizes of the parts, which are necessary for spacing components, they are also necessary for identification purposes. For example, it would be difficult to determine the identity of a certain IC if all the operator had was a group of fourteen pads in two rows.

Once the parts library is created, the parts may be imported into a circuit. During circuit layout, pads are normally used only for jumper wires, since all other pads should theoretically be included in the parts imported from the library. Generally, the parts are laid out first and connected, according to a schematic, with lines of widths chosen from the previously defined set of conditions. The parts are then moved and rotated to take up the least amount of space possible using the least number of jumper wires possible (some were always necessary with one-sided circuit boards), and the corresponding lines are moved, shortened, or lengthened.

The first circuit board I created was the section of the analog to digital converter board containing line drivers and connectors for address input and data output. It was during the layout for this circuit that I had to come up with line widths and pad sizes, so I did some experimenting with different configurations. The actual layout for this section was already provided by my lab focal point, so I was mainly learning the commands and functions of the system. I ran into several mechanical problems with the milling: the milling lines were too deep, giving the conductor lines ragged edges. The holes I used were too large, and the pad apertures too small, leaving only a thin sliver of copper around the hole that would be nearly impossible to solder to. I changed the board around, using pads with smaller, 1.0 mm holes and large conductors around them. I raised the bit so it wouldn't cut so deeply, and I cut out some of the unnecessary lengths of copper so that the board was more compact. I also added a rectangle of milling lines around the entire board so the machine would cut the board out when it was done milling. I drew the cutout lines with a different size milling tool, so that when the program prompted for the second bit, I could just put in the router bit for cutting through the board and the machine would do it automatically.

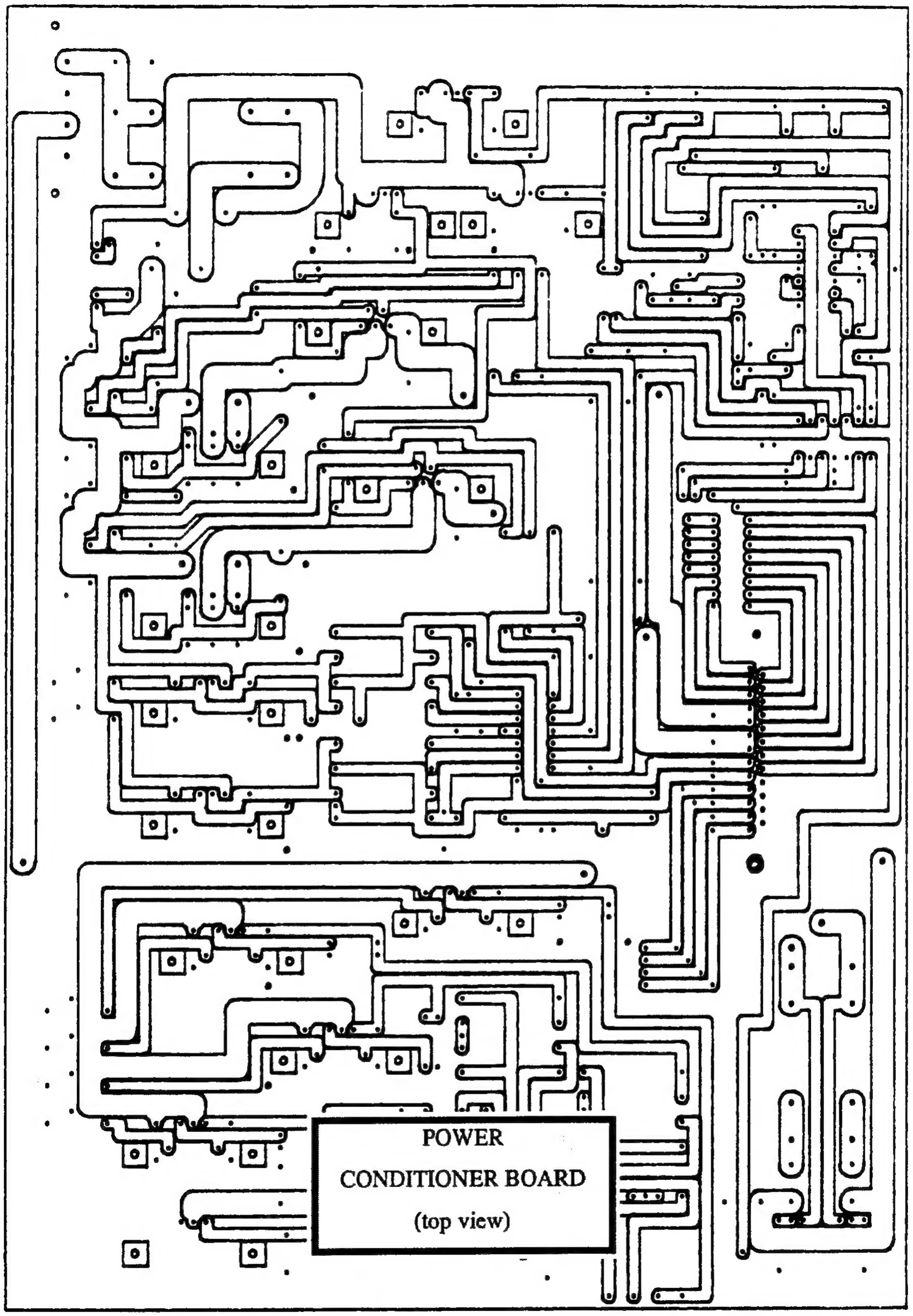
Mechanically, the second board was much better. the only flaw was the new pad apertures, which were slightly too large and overlapped one another. The original circuit design, however, had some connections in the wrong places and had to be redesigned.

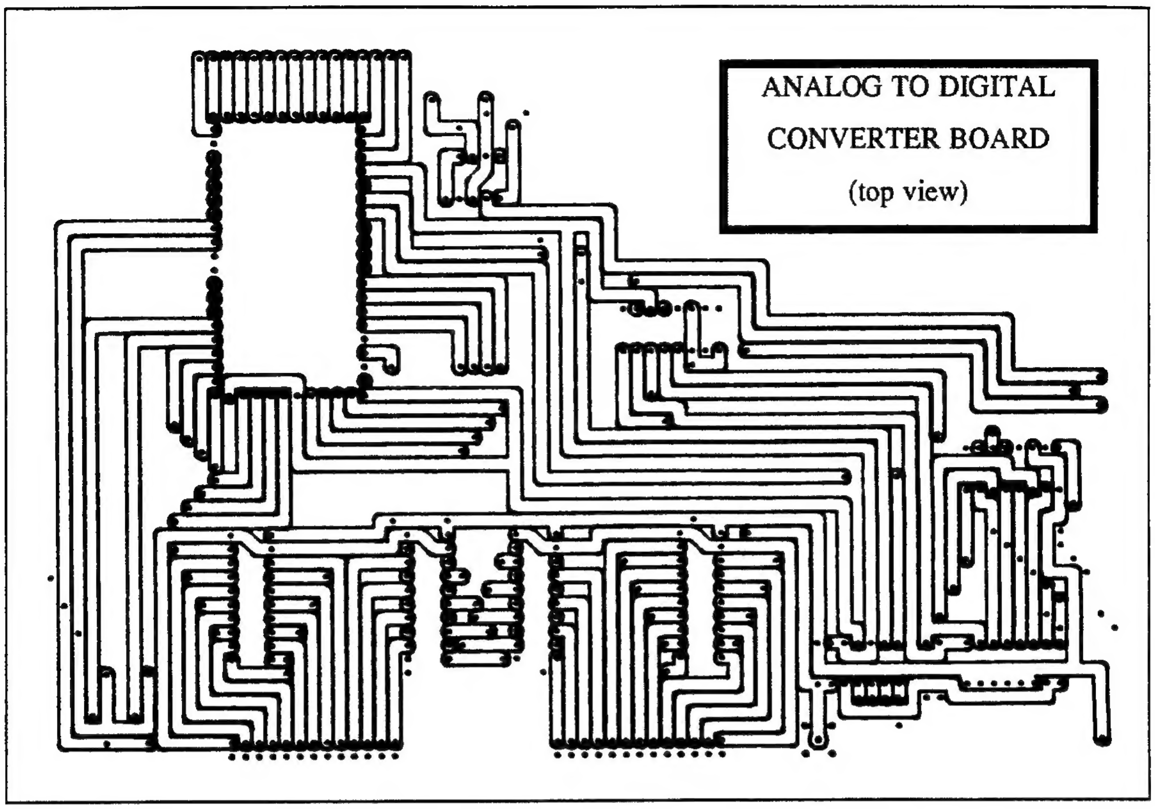
I redrew the new line drivers board design and milled it out. The board came out usable, so I populated it with jumpers, connectors, and sockets. I also constructed a cable to connect it to the Hewlett Packard computer for testing. My lab focal point tested it with both the HP and with an oscilloscope, and it worked fine, so I incorporated it as a part in a new board and started the layout for the A/D converter circuit.

The layout went fairly smoothly, considering I was still learning the software as I was going. It took me three days to design the layout, and two days to solder the components onto it. At first I was very awkward with the soldering iron, but with so much practice I quickly became proficient in soldering. Although I finished the A/D converter printed circuit board, testing was limited to the line drivers. The software on the HP Model 380 was not completed to test the A/D section at the end of my apprenticeship.

The power conditioner section of the circuit was much larger, but by the time I started it I was familiar with most of the features and commands of the CAD software, so the layout went just that much more quickly. The negative voltage section of the power supply board took me about two days to design, and the considerably larger and more complicated positive voltage section took me approximately five days. Extra time was necessary on the positive section because many of the main voltage conductors were too narrow and needed to be widened. I had purposely made the design as compact as possible, so it was necessary to move large portions of the circuit to provide enough space for the wider conductors.

The fabrication of the power conditioner circuit board took a full day, as I had several problems getting the board to fit within the milling machine's limits. When I did finally get it to fit, the board milled out very well. There were several areas where components were too closely spaced, as I discovered during soldering, but the only real design errors involved two jumper wires that I forgot to include, and pins one and two on several of the positive voltage regulators, which were transposed. I soldered for four days and almost completed the board, but several parts that had been ordered had not arrived. I then helped my lab focal point debug what was done of the circuit (we were only missing a few precision resistors and some diodes.) I had made several unintentional solder bridges that shorted adjacent conductors, but when these were cleared up the circuit worked fine (after correction of the transposed pins on the voltage regulators.) I made all of the needed corrections on the CAD file, so the next board constructed will be spaced correctly and have all its pins in the right places. Top views (component side) of the corrected boards are provided on the next two pages. The plots were reduced to seventy or eighty percent of actual size to fit in this report.


POWER
CONDITIONER BOARD
(top view)



Interestingly enough, the first time a plot was attempted of the above circuit board layouts the plotter ground away for five or six hours and had not finished the plot. Perhaps a problem exists in the output of this CAD program also! I immediately consulted the engineer who had patched up the output of the schematic drawing program. Already armed with the first C program to place CRLFs (carriage return line feeds) after any semicolon, we looked at the output file in a text editor. The offending commands were absolute arc commands where a precision of infinity (allowable error of zero) was specified. Apparently the smart plotter spent much time trying to reach the desired precision. A C program was created to post process the output to the plotter. Small arcs (holes) were given twice the error limit as the larger arcs at the end of lines. This program also lowered the pen velocity to 5 from 20, to give a better plot. The circuit board plots were each done in about five minutes with the post processed file from the program below:

```
#include <stdio.h>  /* pcfix2.c    pc-plot 10hour fix*/
#include <string.h>       /* two persons for 5 hours*/
main(argc, argv)
int argc;
```

```
char *argv[];
{
int a, b, c, ch;
unsigned int i;
char line[50];
FILE *fpr, *fpw, *fopen();
   //  fpr = fopen(``testf'',''rt'');
     fpr = fopen(``fixibc'', ``rt'');
     fpw = fopen(``fixibc2'', ``wt'');
while((fgets(line, 49, fpr))!=0){  //ret 0 on EOF
   if((line[0]=='V')||(line[0]=='A')){
if(line[0]=='V') fputs(``VS5;\n'', fpw);
if(line[0]=='A'){
  if(line[strlen(line)-10]=='3'){
 line[strlen(line)-3]='3';
 line[strlen(line)-2]='0';
 line[strlen(line)-1]=';';
  }
  if(line[strlen(line)-10]!='3'){
 line[strlen(line)-3]='1';
 line[strlen(line)-2]='5';
 line[strlen(line)-1]=';';
  }
  fputs(line, fpw);
  fputs(``\n'', fpw);
 }
      }
   else  {
  fputs(line, fpw); }   //end of else
  }//end of while
    //    puts(``''');  //testing only
fclose(fpr);
fclose(fpw);
}
```

Results

The primary result of my summer apprenticeship was that the layout designs and prototype printed circuit boards for the power conditioner and analog to digital interface were completed. These were successfully tested and are now ready for use in the automatic microwave life tester at Rome Laboratory.

A secondary result is that three post processing programs now exist for general use on either of the CAD systems related to circuit board design and fabrication.

Conclusions

It is possible to take a preliminary circuit design, two untested CAD programs, and a computer controlled milling machine and fabricate a couple of working prototype printed circuit boards during an eight week apprenticeship.

Even very expensive commercial software can have minor problems. This was interesting because it was totally unexpected.

It seemed to actually take longer to build something than originally estimated. Even the estimates of experienced engineers were sometimes low.

Once you know that the paper design is correct and build a piece of hardware, new real errors surface in the design. This would indicate that early prototyping is essential.